This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

Anomaly Detection in Real-Time Multi-Threaded Processes Using Hardware Performance Counters

Prashanth Krishnamurthy, Ramesh Karri, Farshad Khorrami

***Abstract*—We propose a novel methodology for real-time monitoring of software running on embedded processors in cyber-physical systems (CPS). The approach uses real-time monitoring of Hardware Performance Counters (HPC) and applies to multi-threaded and interrupt-driven processes typical in Programmable Logic Controller (PLC) implementation of real-time controllers. The methodology uses a black-box approach to profile the target process using HPCs. The time series of HPC measurements over a time window under known-good operating conditions is used to train a machine learning classifier. At run-time, this trained classifier classifies the time series of HPC measurements as baseline (i.e., probabilistically corresponding to a model learned from the training data) or anomalous. The baseline vs. anomalous labels over successive time windows offer robustness against the stochastic variability of code execution on the embedded processor and detect code modifications. We demonstrate effectiveness of the approach on an embedded PLC in a Hardware-In-The-Loop (HITL) testbed emulating a benchmark industrial process. In addition, to illustrate the scalability of the approach, we also apply the methodology to a second PLC platform running a representative embedded control process.**

***Index Terms*—Anomaly Detection, Cyber Security, Programmable Logic Controller, Malware, Resilient Control.**

I. Introduction

Cyber-security for cyber-physical systems (CPS) and industrial control systems (ICS) is becoming increasingly important [1]–[3]. Several attacks on CPS/ICS have been reported [4]–[14]. While general-purpose computer and network security approaches apply to CPS, leveraging the temporal behavior and code structure characteristics of CPS devices can offer complementary solutions. We present a lightweight (i.e., zero-hardware-cost) method for malware characterization and detection in CPS devices using the Hardware Performance Counters (HPCs) digital side channel.

HPCs are special-purpose registers embedded in almost all processors (including Intel, ARM, and PowerPC) and count hardware events such as number of instructions retired, number of branches taken, and other low-level processor events of applications running on the processor. HPCs provide in-depth performance information of software without modifying the source code. The specific HPCs available depends on the particular processor architecture. HPC measurements can be accumulated over the time intervals between successive HPC measurements. A time series of HPC measurements offers a temporal profile of the code being executed. Given a "known-good" embedded device, the HPC time series characterizes the expected temporal characteristics of the code on the embedded processor when it is running the expected code. This paper develops a lightweight method (Figure 1) to detect anomalies using machine learning classification of real-time HPC measurements.

While this implementation considers HPC measurements via software on the embedded device, HPC measurements can also be relayed onto a digital output and remotely collected. This can be done by a kernel module on the device or a dedicated hardware port. One motivation of this study is to demonstrate efficacy of HPC-based monitoring of real-time embedded processes and make a case for hardware support for continuous HPC monitoring in next-generation processors. A monitor can be interfaced to the embedded processor via a dedicated hardware port to enable arms-length monitoring.

A. Key Contributions and Novelty of the Approach

This paper presents real-time anomaly detection in multi-threaded processes (e.g., controller, sensor processing, and sensor fusion implementations) in embedded PLCs. Features extracted from the multidimensional HPC time series of the target process are classified using machine learning to detect mismatches between observed and baseline temporal behavior. Time series of HPC measurements collected from the device under known-good conditions are used without requiring anomalous data. The trained anomaly detector detects malware/modifications that have not been seen before. The approach is applied to controllers running in a PLC in a HITL testbed of an ICS. We show that several modifications can be detected using the approach.

While HPC-based anomaly detection has been considered in prior works (e.g., [15]–[29] – see Section I-B: Related Work), a crucial novelty of the proposed approach is that the primary focus is on real-time multithreaded processes on embedded devices in CPS and enabling continuous monitoring of time series of multi-thread HPC readings of such real-time processes. For this purpose, novel aspects of the proposed approach include:

- An algorithmic methodology that applies to multi-threaded processes wherein the multiple threads could run with vastly different load characteristics and could have a mix of timing-based and event-triggered/interrupt-driven components. Multi-threaded processes are typical in real-time embedded controllers. For example, control

The authors are with the Department of Electrical and Computer Engineering, NYU Tandon School of Engineering, Brooklyn, NY, 11201 USA. e-mails: {prashanth.krishnamurthy, rkarri, khorrami}@nyu.edu.

This work was supported in part by the U.S. Office of Naval Research under Awards N00014-15-1-2182 and N00014-17-1-2006 and by DARPA under AFRL contract FA8750-16-C-0179. The authors are associated with NYU Control/Robotics Research Laboratory (CRRL). Karri is associated with NYU CCS and NYU-AD CCS-AD.

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

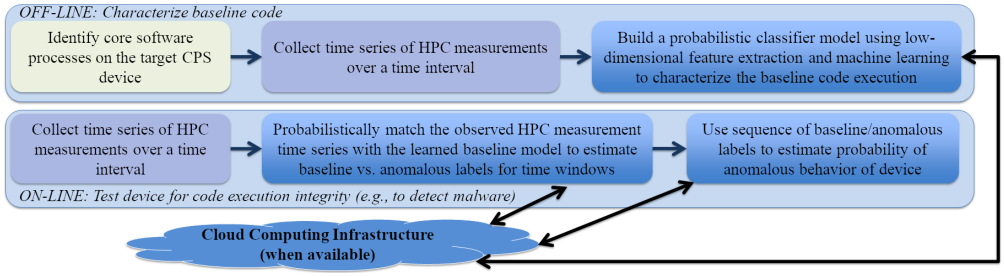


Fig. 1: Overall structure of the machine learning based malware detection and characterization system.

implementations on PLCs typically use separate threads for analog and digital inputs, for network communication with other PLCs, human-machine interfaces (HMI), and user-defined control algorithms.

- An approach to detect malware (or unexpected modifications of the target process) that was not previously seen. The approach does not require malware signatures and uses known-good data from baseline device operation.
- A blackbox ("outside-the-process") approach to real-time monitoring of unmodified processes for which source code is unavailable. HPC measurements are collected by a separate process using kernel-level methods. The monitored process is not instrumented and hence its operation remains unmodified. By using a machine learning based time-series classifier, the approach is oblivious to the structure of the process. The anomaly detector uses time windows of the HPC measurements without assuming timing synchronization and hence does not require temporal alignment of the HPC measurement time series with any internal structure of the process.
- The anomaly detector considers the multi-threaded structure of the target process. The HPCs from each thread are measured separately at each sampling instant to create a vector of HPC measurements for each thread. The matrix generated from these HPC sensor measurement vectors over a sliding window of time as a multidimensional HPC sensor input is used to extract low-dimension features. Feature extraction considers per-thread and cross-thread features. While per-thread features model activity patterns within the threads, cross-thread features model temporal relationships among activity patterns between threads.

B. Related Work

Cyber-security of CPS has been addressed in several works (e.g., [1]–[3] and references therein). Approaches based on monitoring of network traffic have been proposed, for example, in [30], [31]. While [30] addresses monitoring of Modbus/TCP traffic to and from PLCs to detect abnormalities in messages using a deterministic finite automaton model of normal traffic, [31] proposes to monitor BACnet traffic in building automation systems and detect anomalies in time-driven, human-driven, and event-driven network traffic. Unlike network-based anomaly detection methods, the proposed approach is based on processor-level measurements (HPCs) directly on the embedded device and can therefore enable detection of attacks that modify code behavior on the device while not modifying network communication (e.g., changing control algorithm behavior).

As discussed in Section I, HPCs provide a lightweight hardware-level mechanism to monitor applications running on a processor. All modern OSes support HPC-based profilers [20]. Modern Intel processors such as Ivy-Bridge and Broadwell can monitor 468 and 519 performance parameters using HPCs [27]. However, typically only 4 different HPCs (2 on some processors) can be monitored simultaneously.

Reusing HPCs for malware detection was first proposed in [15] in the context of program integrity checking. Malicious program modifications were evaluated at two stages: statically during program load stage and dynamically at runtime. One can use HPCs to detect malware that uses Direct Memory Access (DMA) to attack a host [16] by using HPCs to maintain a record of memory accesses to thereby identify these types of malware. HPCs can be used to detect control-flow integrity attacks [17]. One can use HPCs in a Virtual Machine environment to detect kernel control-flow modifying rootkits [18], [20]. Adaptive compressive sensing can be used to develop a HPC-based malware detection scheme [19]. HPCs were used in [21] to detect Linux and Android malware. This method was further improved in [22] by applying machine learning to detect malware in Windows-based applications such as Internet Explorer and Adobe PDF Reader. A HPC-based approach was developed in [23] for real-time malware detection using behavioral characteristics. A density-based clustering method was proposed in [24] for detection of anomalies in time series of HPC measurements. HPCs have also been used for malicious firmware detection [25] and for intrusion detection in networks [26]. A machine learning based rootkit identification using HPCs was proposed in [27]. In [28], a Support Vector Machine (SVM) based approach was proposed to detect anomalous applications using HPC readings from complete runs of the application (i.e., single set of readings at the end of the application run). In the recent work in [29], the detection of ransomware using HPCs was studied based on autoencoder and Fast Fourier Transform (FFT) methods.

Compared to the prior approaches outlined above, the primary crucial difference of the proposed approach is that we consider embedded devices (e.g., PLCs) running real-time multithreaded processes. This multithreaded aspect is not considered in prior approaches. Furthermore, the proposed approach intrinsically considers continuous monitoring of time series of HPC readings rather than single instantaneous mea-

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

surements or single measurements for entire application runs. The proposed methodology provides a general and scalable framework (based on real-time HPC monitoring of embedded controls processes, multi-dimensional feature extraction over time windows, and anomaly detector training using only data from a good device and without requiring any anomalous data) for continuous monitoring of multi-dimensional data time series arising from real-time multithreaded processes on embedded CPS devices such as PLCs.

Various sources of non-determinacy and randomness due to other running processes, processor features such as out-of-order execution and branch prediction, run-time input signals, and parameters affect the quantitative measurements of HPCs at run-time. The proposed approach accounts for these sources of non-determinacy and randomness by performing probabilistic correlation between the off-line generated baseline models and the real-time constructed models of code executing on a device using robust feature extraction and correlation techniques that operate on time series of HPC measurements over sequences of time windows. Unlike signature-based anomaly detection that relies upon prior knowledge of signatures of specific malware, the proposed approach generates an implicit model of the baseline behavior and detects anomalies relative to this baseline. This enables detection of malware that has not been previously seen by using the run-time characteristics of the monitored code.

C. Paper Roadmap

The considered threat/attack model and the proposed algorithmic framework for HPC-based anomaly detection are described in Section II. Experimental studies of the efficacy of the approach on two embedded PLC devices are presented in Section III. Concluding remarks are provided in Section IV.

II. HPC-Based Anomaly Detection

The attack model and proposed anomaly monitoring approach are illustrated in Figure 2 and discussed below.

A. Threat/Attack Model

An adversary modifies a process on an embedded microcontroller such as in a PLC. For example, the most relevant process in this context will be the process that runs the control logic programmed onto the PLC. This control logic is specified via a Structured Text program or through a graphical Integrated Development Environment (IDE) provided by the PLC vendor and is loaded onto the PLC as an executable binary. Hence, one instance of the attack model considered in this paper is that an adversary gains unauthorized access to a PLC (using a vulnerability in the implementation of the network protocol used for programming the PLC over the network) and replaces the control logic binary on the PLC. Alternatively, an adversary could use a vulnerability in the implementation of a network protocol to remotely access the PLC to modify the control logic process or re-launch it with modifications. Such modifications include changes to command-line parameters to the control logic process, changes to environment variables, and pre-loading of malicious libraries (e.g., using a mechanism such as `LD_PRELOAD` in Linux). The attack is outlined below:

- At some time when an adversary has physical/network access to the embedded device (or its components during manufacturing, integration, or repair), he/she implants a malware on the device or modifies a running process.
- The malware or the implanted modification is activated (e.g., immediately or by a time-based or an event-based trigger) at some time during the operation of the device.

The malware may alter the control behavior (e.g., spoofing readings from a sensor, modifying a variable computed in the control algorithm). The malware may exfiltrate sensitive data from the device. The malware may modify messages being communicated to feed erroneous information to a HMI. Depending on the CPS and the role of the embedded device, the malware may create malicious effects to hamper the function/performance of CPS. As discussed in Section III, the proposed approach provides a methodology for detection of attacks that modify temporal behavior during code execution as seen in time series of HPC readings. Such a change in temporal behavior could be caused, for example, by a modification of a control algorithm by an attacker either by modifying code on the device or by modifying the computation flow through, for example, a buffer overflow based attack.

B. Hardware Performance Counters (HPCs)

HPCs provide measurements (counts) of various hardware-related events during code execution and are used for low-level performance analysis and code tuning. However, HPCs being a hardware-level functionality are an attractive and flexible capability to verify code integrity with negligible performance overhead. HPC-based monitoring can scale to a large class of platforms spanning devices running operating systems (such as Linux) and bare-metal devices.

The HPCs are processor architecture dependent. Example HPCs include: number of instructions executed, processor cycles consumed, interrupts triggered, L1 data cache misses, L1 instruction cache misses and accesses, L1 total (data + instructions) cache misses and accesses, conditional and unconditional branch instructions, and branches that were taken. One cannot read all HPCs at the same time due to hardware constraints. One can time multiplex the HPC measurements, although this has a time overhead due to multiplexing.

The temporal characteristics of device activity during code execution are characterized by total number of events measured by the HPCs, their temporal patterns, and relationships between the monitored events. Sources of non-determinacy will affect the computational characteristics during code execution. Sources of non-determinacy include other active processes, out-of-order execution, branch prediction, run-time inputs, and parameters. To robustly and reliably detect anomalies based on a time series of HPCs, the approach exploits the probabilistic correlation between the baseline behavior and the real-time measurements from the device based on robust feature extraction and correlation. To detect anomalies while being robust to noise, the approach uses temporal patterns seen in a sequence of measurements over a sliding window of time

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

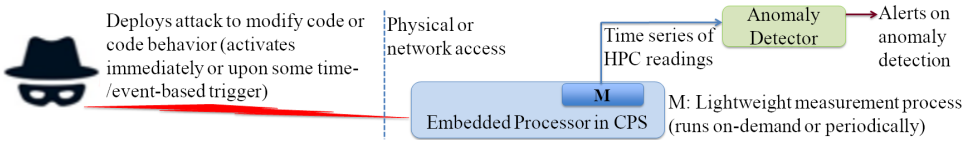


Fig. 2: Threat/attack model and proposed anomaly monitoring approach.

instead of measurements at each time instant. The time series of HPC measurements from a good device is used to learn a model of baseline temporal patterns in the executed code.

C. Collection and Granularity of HPC Time Series

Depending on the device type and application context, HPC monitoring can be defined at various levels of granularity. The "code blocks" can range in granularity from functions to individual processes to the set of all kernel/user-space processes on the device. To address levels of granularity, HPC measurements can be acquired for the entire device, for specific processes therein, for individual threads in a process, or for function libraries or application-specific static and dynamic libraries.

We consider monitoring a specific process (e.g., the critical control logic process on a PLC, which is a particularly relevant application in the context of embedded devices in CPS). The target process will be multi-threaded, as is typical in real-time control logic processes on embedded controllers such as PLCs. HPCs are measured separately for each thread in the multi-threaded process and the anomaly detection uses the multi-dimensional measurement stream of all HPCs separately measured for each thread in the process. There are multiple ways to acquire HPC measurements:

- In-process, by *a priori* instrumenting code of the target.
- Connecting from an external monitoring process according to a fixed sampling rate.
- Hooking into parts of the monitored code (e.g., particular functions) by dynamic instrumentation.

The proposed approach profiles resident threads of an unmodified process in normal operation of the device using the second technique above. Thus, it does not require access to source code or unstripped binaries of the target process. The approach also does not require relaunching the target process or modification of environment variables, and command-line parameters. To simplify describing the approach, a single target process is considered. However, the approach applies to simultaneous monitoring of an arbitrary number of processes. By considering HPC measurements from multiple processes as components within a combined higher-dimensional measurement vector, the feature extraction and anomaly detection can address temporal correlations among activity patterns of different processes. Time series of HPC measurements are collected for the target process under known-good conditions to establish a baseline. When monitoring a device, the observed code execution characteristics are probabilistically matched against expected (baseline) nominal characteristics to detect anomalies. The overall system structure is shown in Figure 1.

To reduce compute load on the embedded device (which is typically constrained in computation capabilities), anomaly detection is carried out by a separate system. The HPCs for the target process are measured by a lightweight, natively compiled, executable (the "measurer") deployed on the target. The HPC measurements are collected to a file and transferred to the analysis system on a separate device, or streamed on-line to the analysis computer. Since the processor in the embedded device is different from the deployment/analysis computer, the lightweight measurer is cross-compiled to a native binary (for the target device) and then transferred. On the embedded device, the measurer can use multiple methods to read HPC measurements for the target process including low-level register access, perf_events or perfctr interfaces in the Linux kernel, high-level PAPI (Performance Application Programming Interface) library, and Intel PCM (Performance Counter Monitor) for Windows and Linux. In this paper, a PLC is used as the representative embedded device and the PAPI library [32] is used to implement the measurer. The system implemented in this paper is described in Section III.

D. Training-Time vs. Run-Time

The malware detector uses the baseline measurements of the target device in a known-good state to train a machine learning model. The approach operates in two stages (Figure 1):

- *Training:* HPC measurements are collected under known-good conditions. To address the various sources of run-time non-determinacy, machine learning is used to learn a probabilistic model of HPC measurement time series corresponding to the baseline.
- *Run-time monitoring:* HPC measurements collected from the device are classified using the trained machine learning classifier to estimate likelihoods that the device is in baseline or anomalous states.

As discussed above, the proposed algorithmic approach is described for a single target process (e.g., a crucial process such as the real-time control logic process on a PLC) but also applies to simultaneous time-correlated monitoring of multiple processes. The target process can be, for example, identified during baseline training and run-time monitoring using the process name and other identifying characteristics. The training-time machine learning and the run-time anomaly detection stages are illustrated in Figure 1.

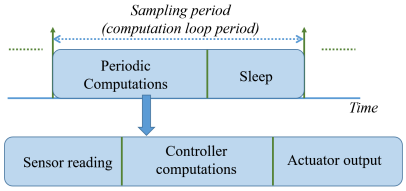
E. HPC Time Series and Problem Formulation

As described above, HPC measurements for the target process are collected as a time series with separate measurements for each thread in the target process. Denoting the number of threads in the target process by n and denoting the HPC measurements for thread i at a particular sampling time t by $m_{i,t}$, the complete measurement vector at time t is of the

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

form $\overline{m}_t = [m_{1,t}^T, \ldots, m_{n,t}^T]^T$. Here, each thread-level HPC measurement $m_{i,t}$ is a vector of length r where r is the number of HPCs being simultaneously measured. The number of HPCs that can be simultaneously measured is processor-dependent and is subject to hardware constraints. Measuring additional HPCs requires multiplexing, which however introduces some overhead. Here, for simplicity, we consider $m_{i,t}$ as corresponding to readings for a set of HPCs that can be simultaneously measured. Hence, the combined measurement $\overline{m}_t$ for the target process is a vector of length nr. At each sampling time t, a combined reading $\overline{m}_t \in \mathcal{R}^{nr\times 1}$ is obtained, thus yielding a time series $\{\overline{m}_{t_1}, \overline{m}_{t_2}, \ldots, \}$ over a sequence of time instants $t_1, t_2, \ldots,$. This sequence of time instants is typically defined based on a fixed sampling rate, i.e., $t_i = t_1 + (i-1)T_s$ where T_s denotes the sampling period. For example, $T_s = 0.001$ s corresponds to a 1 kHz sampling rate, which is typically attainable on embedded processors; on particularly constrained embedded processors, the attainable sampling rate could be lower (e.g., 100 Hz corresponding to $T_s = 0.01$ s).




```
init();
while (1) {
  read_sensors();
  calc_controls();
  write_actuators();
  sleep(T);
}
```

Fig. 3: Control codes in a CPS embedded device have a periodic structure in which a specific set of computations is periodically repeated according to a fixed sampling rate [33]. Each iteration of the periodic loop could, for example, consist of sensor reading, control algorithm computations, and actuator writing steps as in the code snippet shown on the right. In general, these conceptual components/steps could be implemented as combinations of sub-components possibly on multiple threads.

During the training stage, measurements from a known-good device are collected as a time series over a time interval. This sequence of measurements forms the baseline data set. During run-time monitoring, the time series of measurements is the test data set and the problem addressed here is the development of a robust matching approach to decide if the test data set matches the characteristics of the baseline data set or is anomalous. For this purpose, feature extraction algorithms are utilized to extract low-dimensional feature representations from the HPC measurements over time windows. The same feature extraction algorithms are used for both the baseline and the test data sets. A machine learning approach is used to learn a model of feature patterns from the baseline data set. Thereafter, the trained machine learning based system is used to classify the test data set as baseline or anomalous.

A primary motivation in the proposed approach and indeed a central characteristic of embedded CPS devices, which enables the proposed approach to provide robust anomaly detection, is that the typical code structures in such devices have well-defined and typically periodic patterns. Typical implementations (Figure 3) of control logic processes in embedded devices are comprised of periodically repeated iterations of sensor reading, control algorithm computations, and actuator writing steps. Hence, the HPC measurement time series for these processes tends to have approximately periodically repeated patterns although with significant stochastic variations ("noise") due to various non-determinacy effects as discussed in Section II-B.

The algorithm structure is illustrated in Figure 4 and the components are described below.

F. Time Windowing and Low-Dimensional Feature Extraction

The HPC measurements from the target process form a time series of form $\{\overline{m}_{t_1}, \overline{m}_{t_2}, \ldots, \}$. The HPC measurements over sliding windows of time can be used to form various types of time- and frequency-domain feature descriptors [34]–[40]. These feature descriptors are computed for sliding windows of the time series signals and over multiple temporal lengths. Thus, given the time series signal by the set of measurements $\overline{m}_t$, the feature extraction considers time windows of the form $\{\overline{m}_t\}$ for $t \in \mathcal{T}_{(T_1,T_2)}$ where $\mathcal{T}_{(T_1,T_2)}$ denotes all time stamps for the time series signal in the time interval $[T_1, T_2)$. In this time window, features of various types can be extracted from sliding window segments corresponding to time intervals of form $[\tau, \tau+\gamma]$ for a sequence of values of τ and γ. Features can be extracted over sliding window segments (with successive τ being different by smaller than γ) and for different temporal lengths (for different values of γ).

The use of multiple temporal lengths provides a multi-resolution approach that facilitates learning of temporal patterns apparent over different time scales. The possible values of γ are picked to be a discrete set Γ depending on the time scales of the time series signals in the specific application (e.g., depending on the typical control loop sampling periods when monitoring a control logic process, and time scales of local features in the time series signals). Over each time window, multiple types of features are extracted:

- Basic statistics such as max, min, mean, root mean square, variance, skewness, and kurtosis of the measurement data points (HPC samples $\overline{m}_t$) within the time window. These statistics are extracted separately for the different threads and for the different HPC modalities. Statistics such as mean and root mean square characterize levels of activity (within the time window segments and in terms of the different HPC measurement modalities such as number of instructions and number of branches).
- Features based on inter-sample rates of changes. Statistics of inter-sample changes include, for example, the means of absolute values of pair-wise differences of HPC measurements between successive sampling times. The computation of means of absolute values of point-wise derivatives of the time series signal uses three or more successive points for numerical robustness. Statistics of inter-sample changes characterize patterns of time variations of activity (i.e., derivatives of the activity patterns).
- Histogram based methods (e.g., percentage of samples over the mean, percentage of samples in highest 25%).
- Frequency domain methods such as Discrete Fourier Transform (DFT) and Discrete Wavelet Transform (DWT), e.g., frequencies (or mean of frequencies) corresponding to highest few peaks in the DFT. The time-domain and frequency-domain dimensionality reduction

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

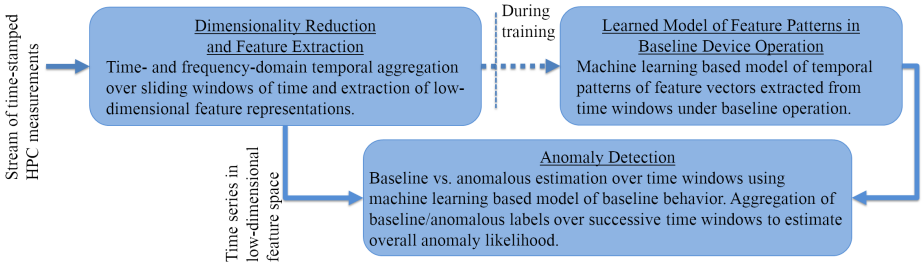


Fig. 4: Algorithmic architecture of the multi-time-scale machine learning based code integrity verification system.

methods provide an information quantization approach to encapsulate time windows of HPC measurements as low-dimensional feature vectors.

- Autocorrelation methods, e.g., lag for which highest autocorrelation is achieved. This feature extracts periodicity characteristics of the time series signal.
- Cross-correlation across threads and across HPC measurement modalities. These features extract characteristics of temporal relationships between activity patterns in different threads and different types of activity patterns.
- Polynomial-based methods, e.g., coefficients of a polynomial representation (e.g., cubic splines, Chebyshev polynomials) computed as closest fit for the time series signal window segment.
- Compression based methods, i.e., a measure of the information content of the signal window segment, e.g., number of bits of most compact representation (to within some approximation threshold). This feature can be computed separately for each thread and/or each HPC modality or can be computed as a combined metric for the multidimensional measurement sequence comprising of HPC measurements from all threads.

Given a time window from which a feature vector is to be extracted, features of the various types described above can be extracted from sliding sub-window segments of different lengths within the time window. The feature descriptions obtained from the multi-resolution hierarchy of time window segment lengths (i.e., for different values of γ) are aggregated by concatenating with the features extracted from the sliding window segment of the top-level γ value). Combinations of low-dimensional feature descriptors essentially provide semantic hashes comprising of low-dimensional feature representations of the measurements over time windows. Using these feature extraction techniques, the HPC measurement set over a time window is mapped to a feature vector. This feature vector is the input to a machine learning based classifier.

G. Machine Learning Based Anomaly Detection

The baseline vs. anomalous classification entails learning an implicit model of the characteristics of the feature patterns corresponding to baseline behavior and using the learned model to evaluate a data set obtained at run-time. While there are several linear and nonlinear classification approaches, an SVM approach provides a general-purpose methodology that can yield robust classification accuracy. In contrast with linear classifiers, an SVM accurately fits complex nonlinear separating surfaces between different categories of data by using a nonlinear kernel (such as the radial basis function) to classify data in a high-dimensional feature space. The SVM uses the *kernel trick* to compute dot products in an implicit high-dimensional feature space through a nonlinear function in the lower-dimensional input space.

We use the one-class SVM approach to model baseline behavior and detect anomalies that deviate from it. The SVM is trained based on "good" data. The trained SVM, when presented a new input, determines if it is from the same class as the training data or is "different." There is only one class that the SVM is ever trained for, i.e. the baseline. The SVM is not trained for any anomalous conditions. At run-time, it decides if the data is similar to the baseline for which it is trained or if it appears different. This approach is ideal for run-time monitoring since only "good" operational data is available in a typical application in an embedded CPS device. Malware/modifications are unexpected and not necessarily known *a priori* (i.e., no specific known signatures). By learning characteristics of the baseline and determining if the observed data differs from this baseline is an effective way to detect modifications that have not been previously seen.

As described in Sections II-E and II-F, the input to the SVM is a feature vector extracted from a time window. Using the notation introduced in Section II-E for the HPC measurements, sliding time windows are defined to be of the form $\{\overline{m}_t | t \in [T_0 + iT, T_0 + iT + \overline{T})\}$ where T_0, T, and $\overline{T}$ are constants and $i = 0, 1, 2, 3, \ldots$, T_0 is an initial time, T is the shift between successive time windows, and $\overline{T}$ is the length of each time window. T is picked to be substantially smaller than $\overline{T}$ to create a substantial overlap between successive time windows. This increases the robustness of the approach compared to a sequence of non-overlapping windows for two reasons. First, the number of time windows is much larger when overlapping is allowed; therefore, improving robustness to occasional misclassification errors. Second, subtle temporal patterns can be learned (during training time) and discriminated (during run-time monitoring) by extracting features and performing anomaly detection classifications over several sliding partially overlapping time windows. Example feature extractions are shown in Figure 5.

While the framework can support a wide range of feature types, each application would require only a small subset of the feature types. Identifying the set of feature types that are effective for an application entails analyzing the typical variations and repeatability of the features when operating on

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

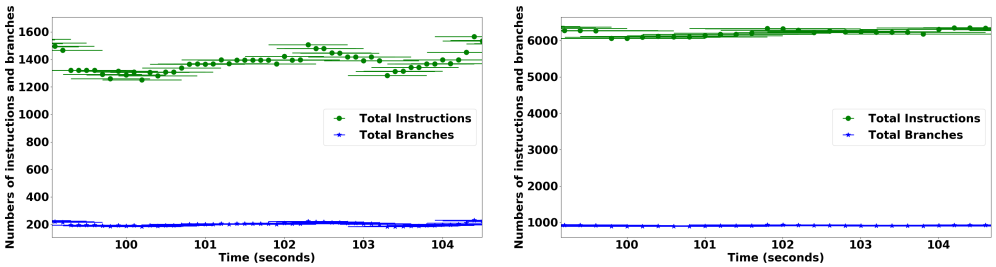


Fig. 5: Examples of feature extraction from sliding windows of time series of HPC measurements. The time series corresponds to the PLC_Task thread that executes a user-loaded Structured Text code within the multi-threaded PLC process on the Wago PLC. The left-side plot shows the averages of numbers of instructions and branches for overlapping sliding time windows of length 1 second. The right-side plot shows the root mean squares of numbers of instructions and branches for overlapping sliding time windows of length 2 seconds. The green dots and blue *'s indicate the computed features for numbers of instructions and branches, respectively.

time windows of baseline data. The feature types that are most effective for an application are ones that exhibit some, but not too much, variation over the baseline data so that one can verify that those feature types are sensitive to code execution while being somewhat repeatable during baseline operation. The choice of feature types is not critical as long as a variety of features are used. Addition or removal of a feature does not critically affect classification performance due to the robust nonlinear classification capabilities of a one-class SVM.

When training the one-class SVM, the set of feature vectors obtained from a baseline data set is used as the training data. During run-time monitoring, each such feature vector is classified by the trained SVM as non-anomalous (i.e., baseline) or anomalous. Thus, given a measurement stream at run-time, the sequence of (partially overlapping) sliding time windows yields a sequence of SVM-based classification labels as baseline or anomalous. Thereafter, temporal aggregation is used to remove any intermittent misclassification errors (i.e., false negatives when a time window from baseline conditions is classified as anomalous; and false positives when a time window under an actual anomaly is classified as baseline). We define positives and negatives in the context of the machine learning classification as labeling as baseline and anomalous, respectively. The simple temporal aggregation given by majority voting over a sequence of time windows (e.g., N_w time windows) typically suffices. Due to the partial overlapping structure of the time window decomposition, a sequence of N_w time windows is of length $(N_w - 1)T + \overline{T}$, and not $N_w\overline{T}$, which it would have been if successive time windows were required to be non-overlapping. Since T is much smaller than $\overline{T}$, $(N_w - 1)T + \overline{T}$ is much smaller than $N_w\overline{T}$. This highlights another advantage (temporal granularity) of sliding time window based feature extraction and classification.

III. Experimental Studies

The approach is demonstrated below on an embedded PLC within a HITL testbed modeling an ICS benchmark process.

A. HITL Experimental Testbed

Tennessee Eastman (TE) process: The TE process is a complex open-loop unstable industrial process benchmark modeling the dynamic behavior of a real chemical process [41], [42]. The chemical process has five operation units – reactor, product condenser, vapor-liquid separator, compressor, and stripper. In the TE process, gaseous reactants A through E are combined to produce two liquid products, G and H, and one liquid byproduct F according to a set of chemical reactions. The TE process represents a real-world industrial process with 50 states, 41 measured variables with Gaussian noise components, 12 manipulated variables, and 13 disturbance signals that can be applied to the system.

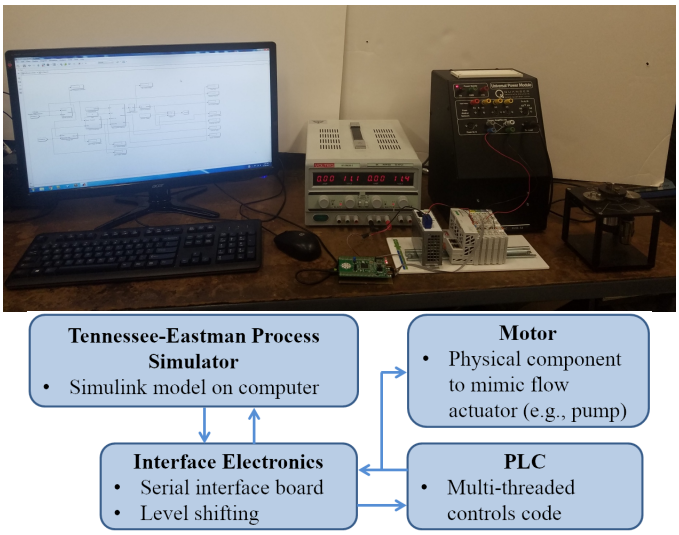


Fig. 6: The HITL testbed [42], [43] includes the TE simulator, Wago PLC controller, and a DC motor mimicking a flow actuator in the TE process. The top picture shows the testbed and the bottom picture shows a block diagram of the testbed.

The HITL testbed [42], [43] shown in Figure 6 includes the the TE process implemented as a Matlab/Simulink model running on a PC. The PC is connected via a custom built interface board to a Wago PLC. Two of the critical control loops of the TE process are implemented on the PLC. The control loops on the PLC form the aggregate function of two cascaded PI-loops which control the reactor pressure and the reactor purge rate. The interface board includes electronics for voltage level-shifting, serial interface, and digital-to-analog and analog-to-digital conversion to interface the PC with the analog input/output of the PLC. An ethernet connection between the PC and the PLC is used to re-program the PLC via

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

Wago CoDeSys integrated development environment (IDE). A rotary servo motor (with gear train) is connected to the output of the PLC to mimic a physical flow actuator (e.g., a pump) in the industrial process. Run-time monitoring is implemented on the PLC.

B. Embedded Device: Wago PLC

A Wago PLC [44] is used as the embedded CPS device. As described in Section III-A, the Wago PLC is integrated into a HITL testbed that models an ICS process and implements two control loops in the closed-loop process. The Wago PLC contains an ARM Cortex A8 processor running a custom real-time Linux distribution. The control logic for the PLC is specified using a Structured Text program (IEC 61131-3), which runs on the PLC as a multi-threaded process (23 threads - for I/O, controls, handling of events with different priorities). The Structured Text code is compiled into a binary via CoDeSys IDE. This binary is transferred to the PLC and is loaded using pre-installed binaries on the PLC.

For run-time monitoring of the control logic process running on the PLC, a separate light-weight measurer process is deployed as described in Section II-C. Using the PAPI library [32], the measurer process remotely reads HPC measurements corresponding to the target process (which is specified to the measurer using the process name). HPC measurements are collected at a fixed pre-specified sampling rate (e.g., 1 kHz). The light-weight measurer is implemented in C++ and is cross-compiled (using the ARM gcc cross-compiler). The compiled measurer binary is loaded to the PLC via ssh and is then started from the command line. The HPC measurements can be either logged to a file for a specified time interval (and then remotely processed) or directly streamed over ssh to the anomaly detection program running on a separate computer.

The ARM processor in the PLC provides several HPCs including number of instructions, number of branches, number of cycles, and number of L1 instruction cache misses. However, only two HPCs can be measured simultaneously. While one can time-multiplex to read more than two HPCs, there are overheads associated with multiplexing. To study the efficacy of the proposed approach in Section III-D, we consider two HPCs: number of instructions and number of branches. These are among the most common types of HPCs and are available in all modern processors. We will see in Section III-E that a different choice of HPCs (number of processor cycles and number of L1 cache misses) can also be used and comparable anomaly detection performance is obtained as in Section III-D. In general, as noted in Section III-E, different combinations of HPCs can be used as long as the HPCs chosen are reasonably indicative of run-time processor activity during code execution. Numbers of instructions, branches, cycles, cache misses, etc., are all very relevant in this context. A constraint however is that only a fixed number of HPCs can be monitored simultaneously, therefore necessitating a choice of HPCs to read. In general, in a continuous monitoring context, switching periodically between different combinations of HPCs can also be utilized to increase probability of anomaly detection. HPCs are measured separately for each thread in the target process (see Section II-C). Hence, with the two HPCs, the HPC measurement vector at each sampling instant is a vector of length $2n$ where n denotes the number of threads in the target process. From a time window of measurement vectors, feature descriptors are extracted (see Section II-F) and a one-class SVM classifier is trained (see Section II-G).

C. Sample Controller Code and Example Modifications

A proportional-integral-derivative (PID) controller implemented as part of a control loop on the Wago PLC in the TE HITL testbed is shown in the example Structured Text control logic code snippet in Figure 7. The Structured Text code is compiled into a native binary and loaded onto the PLC using the CoDeSys IDE. HPC measurements are collected for this binary when running on the PLC along with pre-installed binaries on the PLC.

```
PID_INPUT := WORD_TO_REAL(INPUT)/ANALOG_SCALING;
PID_LOOP(
        ACTUAL    := PID_INPUT,
        SET_POINT := PID_SETPOINT,
        KP        := PID_KP,
        TN        := PID_KI,
        TV        := PID_KD,
        Y_MANUAL  := 35,
        Y_OFFSET  := 0,
        Y_MIN     := PID_OUTPUT_MIN,
        Y_MAX     := PID_OUTPUT_MAX,
        MANUAL    := FALSE,
        RESET     := FALSE,
        Y              => PID_OUTPUT,
        LIMITS_ACTIVE  => PID_LIMITS_ACTIVE,
        OVERFLOW       => PID_OVERFLOW);
OUTPUT := REAL_TO_WORD(PID_OUTPUT*ANALOG_SCALING);
```

Fig. 7: Control logic in Structured Text. An analog input `INPUT` is processed by a PID controller to generate an analog output value `OUTPUT`. The variables `PID_KP`, `PID_KI`, and `PID_KD` are proportional, integral, and derivative gain parameters of the controller.

Following are six representative modifications of the Structured Text code reflecting practically relevant attacks:

- A_1: override input (by introducing a line of code to override value of the input to the PID code, i.e., `INPUT`).
- A_2: saturate input (by introducing two lines of code with an `IF` condition on the input value).
- A_3: disable PID control code (by commenting PID block).
- A_4: modify PID code to "manual" mode (i.e., fix the output value to a constant).
- A_5: modify PID controller structure as a cascade of two PID controllers (i.e., by sending output of a PID controller to a second PID controller and using the output of the second one as the overall controller output).
- A_6: override output (by adding a line of code to override value of the output of PID code, i.e., `OUTPUT`).

HPC readings for numbers of instructions and branches for the baseline (unmodified) code are shown in Figure 8 and HPC readings under each of the modifications $A_1, \ldots, A_6$ are shown in Figure 9. The modifications are small (e.g., one extra line in A_1 and A_6). Hence HPC time series for baseline and modifications are similar in macroscopic aspects. The intermittent spikes in HPC measurements are due to non-deterministic

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

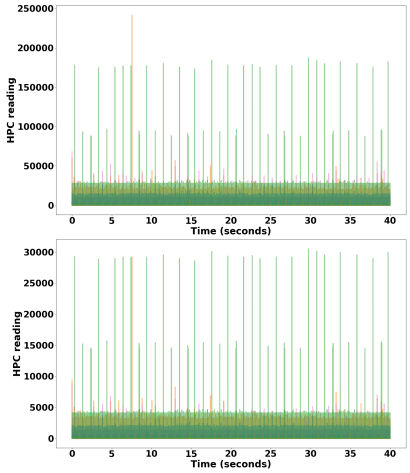


Fig. 8: Time series of HPC measurements for the multi-threaded controller process on the Wago PLC when running the baseline (unmodified) code. The top and bottom plots show the HPC measurements for number of instructions and number of branches, respectively. The HPC measurements are collected at 1 kHz sampling rate for the 23 threads in the process. The readings for each thread are shown in a different color. It is seen that although there are approximately periodic patterns within the time series, there is also a significant amount of stochastic variability and intermittent spikes.

effects discussed in Section II-B and cannot reliably be used to distinguish baseline and anomalies. A robust and accurate classification has to rely upon subtle temporal patterns in the time series of HPC measurements. Hence, feature vectors are extracted from sliding time windows as discussed in Section II-F and used for SVM-based classification.

D. Results of Proposed Anomaly Detection

Using the notation in Section II-G, a sequence of sliding time windows is used as $\{\overline{m}_t | t \in [T_0 + iT, T_0 + iT + \overline{T})\}$ with $T = 0.025$ s, $\overline{T} = 0.25$ s. Since the process has 23 threads and two HPCs are measured, the measurement vector at each sampling instant is of length 46. The measurements are collected at a sampling rate of 1 kHz. The values of T and $\overline{T}$ correspond to a sequence of sliding time windows with each time window 250 samples long and with a shift of 25 samples between successive (partially overlapping) time windows.

From each time window, features are extracted using the techniques in Section II-F. As discussed in Section II-G, while a wide variety of feature types can be used, an adequate set of features is somewhat application-dependent. Based on analysis of variations and repeatability of various feature types for baseline data as discussed in Section II-G, a small set of feature types was defined for this application. In particular, the feature vector was defined to include means, standard deviations, means of inter-sample changes, and percentages over median over the time window for each HPC modality and each thread. These features are computed separately for each of the HPC modalities and each thread. The feature vector was simplified by considering a subset of threads (7 threads) that exhibited some level of activity (to ignore quiescent background threads). The cross-correlation with zero lag (after subtracting the means) of the HPC measurements corresponding to numbers of instructions for two of the threads (the PLC_Task thread and the bus cycle task thread) was also included in the feature vector. Variations of this feature vector were also considered. As long as a sufficient variety of feature types is utilized, small changes of feature vector composition do not significantly impact classification performance.

HPC measurement data sets were collected from the Wago PLC over a time interval of 300 seconds for the baseline code and for each of the six modifications described in Section III-C. From the baseline code, the first 75% was utilized as the training data for the one-class SVM. Hence, with the 1 kHz sampling rate utilized for the HPC readings, the training data comprises of 225,000 samples. As discussed above, the SVM based classification operates over time windows that are 250 samples long with a shift of 25 samples between time windows. Hence, the training data provides 8991 time windows. At each time instant, the measurement vector comprises of 46 HPC measurements (readings of two HPCs from each of the 23 threads in the PLC process). Hence, each time window comprises of a measurement matrix of dimension 46×250. From these time windows, feature vectors are extracted as discussed in Section II-F with the specific feature vector composition defined above. The remaining 25% of the baseline data set and all the data sets for the six modifications are utilized as the test data sets. Both the baseline and the anomaly data sets are collected from the running multi-threaded controller process on the PLC. The measurer executable is deployed after the PLC is fully booted up and starts monitoring the already running PLC process. As seen in Figures 8 and 9 in Section III-C, the HPC time series exhibit approximately periodic patterns but with significant stochastic variability and intermittent spikes.

A radial basis function was utilized as the kernel for the SVM. As discussed in Section II-G, the training of the one-class SVM utilizes only the baseline data and does not utilize any data from anomalous code execution. The one-class SVM is an unsupervised algorithm that learns the distribution of the provided data to enable outlier/novelty detection, i.e., essentially learns a representation based on the feature vectors in the training data set so as to enable classifying test data to similar to training data (i.e., normal) vs. different from training data (i.e., novel or anomalous). For this purpose, the SVM internally uses a hinge loss function (that penalizes misclassifications) along with a regularization term. To trade off between overfitting vs. generalization during training (i.e., effectively trading off false positives and false negatives of the trained classifier), the one-class SVM uses two crucial parameters γ_{svm} and ν_{svm}. The parameter γ_{svm} controls the range of influence of each training data point selected as a support vector and intuitively, larger values of γ_{svm} tend to increase the complexity of the decision surface and tend towards overfitting. The parameter ν_{svm} indicates the percentage of outliers in (noisy) training data and decreasing ν_{svm} tends towards overfitting. Values $\gamma_{svm} = 0.15$ and $\nu_{svm} = 0.001$ were found to be, in general, a good choice of parameters over a wide range of scenarios (different HPCs, different features, and even different PLCs). Since only baseline data was used for training, an explicit cross-validation was not used. If some

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

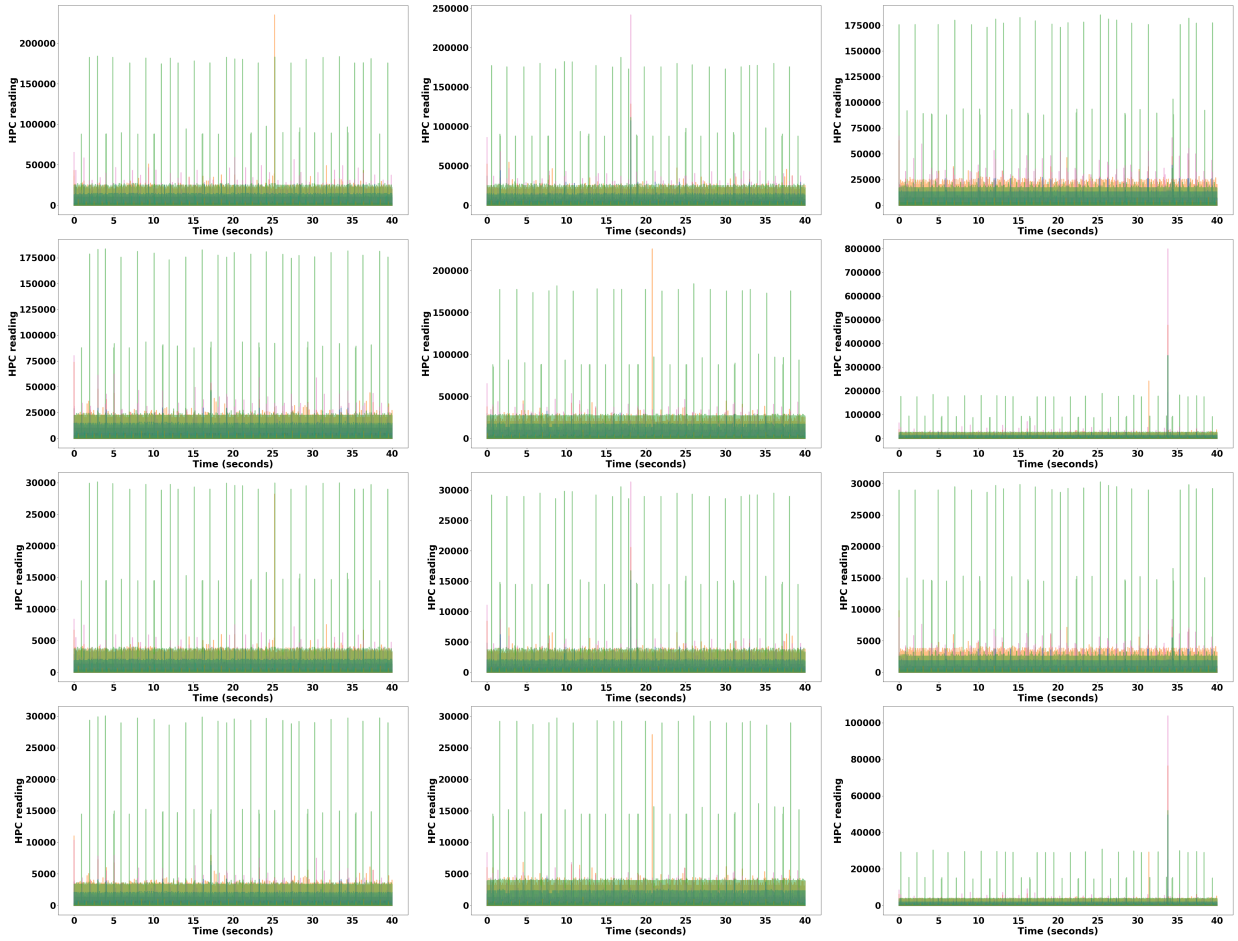


Fig. 9: Time series of HPC measurements for the multi-threaded controller process on the Wago PLC for each of the six attack types considered ($A_1, \ldots, A_6$). The top two rows show the HPC measurement time series for number of instructions for the six modifications $A_1, \ldots, A_6$. The bottom two rows show HPC measurement time series for number of branches. The HPC measurements are collected at 1 kHz sampling rate for the 23 threads in the process. The readings for the different threads are shown in different colors as in Figure 8.

anomaly data is assumed to be available during training, a mix of baseline and anomalous data can be used for cross-validation to tune the SVM parameters.

The trained SVM was tested on the test data sets. The anomaly detection results over the sliding time windows are illustrated in Figure 10. The points in the plots in Figure 10 correspond to the SVM-based classification for feature vectors extracted from the sequence of partially overlapping sliding time windows (the classification for each time window is represented by a point at the end of that time interval in the plots in Figure 10). The trained SVM was 100% accurate when detecting modifications A_1, A_2, A_3, A_4, and A_6 as shown in Table I. Hence, plots for these modifications are not shown in Figure 10. Accuracy of the anomaly detection for the baseline test data set and data sets for modifications are shown in Table I. In Table I, entries that do not have any number are marked '-'. For example, false positives are not relevant when classifying a baseline test data set. The plot for the baseline data set in Figure 10 starts at time $t = 225$ s since the data set before this is used for training. None of the data sets for anomalous conditions are used when training the classifier.

The anomaly detection results in Figure 10 are for each time window and are not a temporal aggregation over a sequence of sliding time windows. By taking sequences of sliding time windows into account to determine baseline vs. anomalous labels, and by temporally aggregating (via majority voting) over a sequence of sliding time windows to mitigate sparse misclassification errors, near 100% accuracy is achieved as shown in Table II. In Table II, the parameter N_w is set to 20 (i.e., sequences of 20 partially overlapping sliding time windows are used to emit baseline/anomalous labels as discussed in Section II-G). The choice of the parameter N_w for majority voting is, in general, a trade-off between

TABLE I: Anomaly detection accuracy over sliding time windows of baseline/anomalous data. Percentages of true positives, false positives, true negatives, and false negatives in the classification are summarized. Here, classifications as baseline and anomalous are considered as positives and negatives, respectively.

Test data set	True positives (%)	False positives (%)	True negatives (%)	False negatives (%)
Baseline	96.73	-	-	3.27
A_1	-	0	100	-
A_2	-	0	100	-
A_3	-	0	100	-
A_4	-	0	100	-
A_5	-	2.39	97.61	-
A_6	-	0	100	-

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

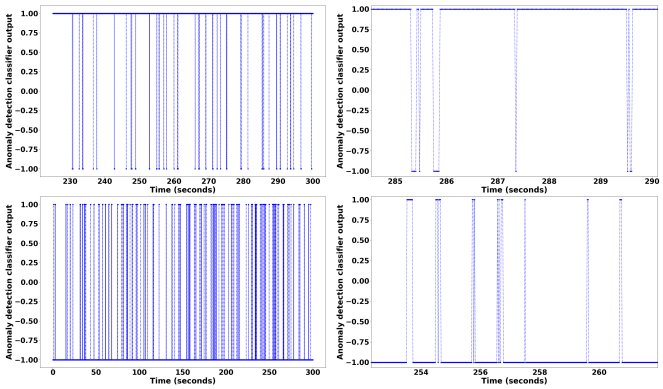


Fig. 10: Anomaly detection over sliding time windows (without majority voting over sequences of time windows). The first row corresponds to anomaly detection in test data set from baseline operation and the second row corresponds to a test data set for modification A_5. In each plot, 1 and -1 indicate that the classifier generated an estimate of baseline or anomalous, respectively, given a sliding time window of data ending at that time instant. In the first row, points which are -1 indicate mis-classifications and in the second row, points which are at 1 indicate mis-classifications. The right-side figures in each row show a zoomed-in view to visualize mis-classification.

temporal granularity (i.e., pinpointing specific time instants where anomalous behavior is seen) and anomaly detection accuracy (both in terms of false positives and false negatives). With, for example, $N_w = 20$, the total aggregated time window over which the majority voting is performed is of length $19 \times 25 + 250$ samples, since each time window for classification is of length 250 samples and the time shift between the overlapping time windows is 25 samples. Therefore, the aggregated time window over which the majority voting is performed corresponds to 0.725 seconds with the 1 kHz sampling rate used here. In comparison, a choice of $N_w = 40$ corresponds to 1.225 seconds. It is to be noted that since successive time windows are overlapping and the time shift (T) between time windows is much smaller than the length ($\overline{T}$) of the time windows, even an intermittent anomaly of a short time duration will be seen in multiple successive time windows making such an anomaly detectable even with majority voting. To convert the labels $l_1, \ldots, l_{N_w}$ over a sequence of N_w overlapping time windows to a binary baseline/anomalous label, a simple majority is used based on defining baseline and anomalous as +1 and -1, respectively, and defining the aggregate label as the sign of $\sum_{i=1}^{N_w} l_i$.

TABLE II: Anomaly detection accuracy over sliding time windows of baseline/anomalous data with majority voting over $N_w = 20$ time windows used to assign baseline vs. anomalous labels for a sliding set of 20 time windows.

Test data set	True positives (%)	False positives (%)	True negatives (%)	False negatives (%)
Baseline	100	-	-	0
A_1	-	0	100	-
A_2	-	0	100	-
A_3	-	0	100	-
A_4	-	0	100	-
A_5	-	0.2	99.8	-
A_6	-	0	100	-

The accuracy of anomaly detection for different settings of N_w are summarized in Table III. The *precision* is given as $P = \frac{p_t}{p_t+p_f}$ where p_t is the number of true positives (i.e., time windows that correspond to baseline and classified as baseline) across all the test data sets and p_f is the number of false positives (i.e., time windows that correspond to anomaly and are classified as baseline). The *recall* is given as $R = \frac{p_t}{p_t+n_f}$ where n_f is the number of false negatives (i.e., time windows that correspond to baseline classified anomalous). $F_1 = \frac{2PR}{P+R}$ is the harmonic mean of the precision and recall scores. Since only 25% of the baseline test data was used for testing (with the remaining 75% used for training), the baseline test data was replicated four times in the computation of the numbers for precision, recall, and F_1 in Table III, i.e., the baseline test data and each of the attack test data ($A_1, \ldots, A_6$) were considered as being for the same time interval of 300 seconds.

TABLE III: Anomaly detection accuracy for varying lengths of sequences of time windows used for baseline vs. anomalous labels.

N_w	Baseline data classification accuracy (%)	Anomalous data classification accuracy (%)	Precision	Recall	F_1 score
1	96.73	99.60	0.976	0.967	0.972
10	98.73	99.87	0.992	0.987	0.990
20	100	99.97	0.998	1.0	0.999
40	100	100	1.0	1.0	1.0

E. Analyses with different configurations of HPCs and multi-thread features

In the above results, the HPC readings for numbers of instructions and branches were utilized along with the feature vector defined above in Section III-D. As discussed in Section II-B, multiple HPCs are typically available on current-day processors, with however a constraint on how many HPCs can be read simultaneously. In the ARM Cortex processor used in the PLC considered here for experimental studies, two HPCs can be read simultaneously. To evaluate the performance of the proposed anomaly detection approach when a different combination of HPCs is selected for monitoring, the HPC readings corresponding to numbers of processor cycles and L1 cache misses were collected when running each of the test codes as discussed above (baseline and attacks $A_1, \ldots, A_6$). The feature vector was defined to be analogous to Section III-D, i.e., means, standard deviations, means of inter-sample changes, and percentages over median over the time window for each HPC modality and each of the 7 non-quiescent threads and the cross-correlation with zero lag (after subtracting the means) of the HPC measurements corresponding to the first HPC modality (i.e., number of processor cycles) for the PLC_Task thread and the bus cycle task thread. With this feature vector and with the same SVM training procedure as discussed above in Section III-D, the anomaly detection results are summarized in Table IV.

TABLE IV: Anomaly detection accuracy for varying lengths of sequences of time windows used for baseline vs. anomalous labels; using HPCs for numbers of processor cycles and L1 cache misses.

N_w	Baseline data classification accuracy (%)	Anomalous data classification accuracy (%)	Precision	Recall	F_1 score
1	97.36	100	1.0	0.974	0.987
10	98.66	100	1.0	0.987	0.993
20	99.30	100	1.0	0.993	0.996
40	100	100	1.0	1.0	1.0

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

To further study the performance of the proposed approach with variations in the feature vector composition, the feature vector was changed to not include the cross-correlation (with zero lag) between numbers of processor cycles of the PLC_Task thread and the bus cycle task thread. Instead, the feature vector was defined to include the cross-correlations with lag of 0.05 s of each of the following pairs of signals: numbers of processor cycles of the PLC_Task thread and the bus cycle task thread, numbers of L1 cache misses of the PLC_Task thread and the bus cycle task thread, numbers of processor cycles of the PLC_Task thread and the asynchronous bus event thread, numbers of L1 cache misses of the PLC_Task thread and the asynchronous bus event thread. With this modified feature vector and with the same SVM training procedure as in Section III-D, the anomaly detection results are summarized in Table V.

TABLE V: Anomaly detection accuracy for varying lengths of sequences of time windows used for baseline vs. anomalous labels; using HPCs related to numbers of processor cycles and L1 cache misses and with modified feature vector.

N_w	Baseline data classification accuracy (%)	Anomalous data classification accuracy (%)	Precision	Recall	F_1 score
1	96.40	100	1.0	0.964	0.982
10	97.89	100	1.0	0.979	0.989
20	99.26	100	1.0	0.993	0.996
40	100	100	1.0	1.0	1.0

The anomaly detection performance in Table IV is comparable to Table III indicating that the HPCs related to numbers of processor cycles and L1 cache misses are also as effective as the HPCs related to numbers of instructions and branches for the purpose of anomaly detection. Also, the anomaly detection performance in Table V is comparable to Table IV and Table III indicating that small variations in the feature vector composition do not significantly impact the classification performance. This highlights the intrinsic robustness of HPCs and the proposed methodology in the sense that variations in the specific choice of HPCs and features do not significant impact performance as long as the HPCs are indicative of run-time activity levels (e.g., instructions, branches, processor cycles, cache misses) and the feature vector provides temporal multi-thread aggregations with reasonable variety.

F. Experimental Studies on a Second PLC

To study the generality and scalability of the approach, the methodology is applied to another PLC (a Raspberry Pi running the OpenPLC open-source PLC system [45], [46]). The underlying software structure of the target process (real-time PLC process) and hardware platform are quite different from the Wago PLC. Specifically, the OpenPLC web-based front-end translates Structured Text code into C code and invokes a compiler on the embedded device to generate a new OpenPLC binary. The OpenPLC binary on the device has 3 threads unlike the 23 threads on the Wago PLC. The processor (a quad-core ARM Cortex) on the Raspberry Pi allows monitoring of 6 HPCs simultaneously instead of 2 as in the Wago PLC. However, inspite of these differences, it is seen that the proposed algorithmic methodology is directly applicable to the new PLC platform and the effort for porting to a new PLC is essentially simply related to setting up the deployment process (software deployment access method, cross-compiler tool chain, etc.). HPCs corresponding to numbers of instructions, branches, stores, cycles, L1 instruction cache misses, and L2 data cache misses are used for monitoring. Of the 3 threads in the OpenPLC process, one of the threads was quiscent. The feature vector for the one-class SVM was defined to include: (a) means, standard deviations, means of inter-sample changes, and percentages over median over the time window for each HPC modality for each of the 2 non-quiescent threads; (b) cross-correlations with 0 lag and with 0.05 s lag between the HPC measurements (for each HPC modality) for the 2 non-quiescent threads. With the same SVM training procedure as in Section III-D, the anomaly detection results are summarized in Table VI for baseline code being a PID and anomalous code modification being a saturation of the sensor input and actuator output values. As noted in Sections III-D and III-E, different HPCs and features can also be used without significant changes in accuracy.

TABLE VI: Anomaly detection accuracy on the second PLC (Raspberry Pi running OpenPLC) for varying lengths of sequences of time windows used for baseline vs. anomalous labels.

N_w	Baseline data classification accuracy (%)	Anomalous data classification accuracy (%)	Precision	Recall	F_1 score
1	80.41	87.49	0.864	0.804	0.833
10	85.55	88.86	0.883	0.855	0.869
20	87.48	90.54	0.901	0.875	0.888
40	93.19	93.11	0.930	0.932	0.931

IV. Concluding Remarks

A framework for real-time monitoring of processes on embedded CPS devices was presented. The approach uses machine learning classification of time series of HPC measurements. The approach applies to multithreaded processes in real-time control implementations. Some of the salient aspects of the proposed approach and the applicability to a range of scenarios are discussed below.

- The methodology **targets code running on embedded devices** (e.g., control and sensor processing algorithms). The code modifications considered are persistent and change some aspect of the operation of the original code (e.g., change a control algorithm). Furthermore, while the proposed approach is primarily aimed at detecting changes in code execution patterns on the embedded device, the proposed anomaly detection approach can also potentially detect network traffic based attacks such as DDoS (distributed denial of service) if such attacks excite code execution patterns different from the baseline.
- **Application-specific processes are monitored** (e.g., the main control process). Depending on the application, more processes could be monitored. In the limit, one can monitor all processes in a round-robin manner in batches.
- While the **monitored code is nominally periodic**, the specific periodic code could vary over time (e.g., depending on some operational state). Hence, to detect anomalies relative to a baseline, the training data should capture the operating modes/stages of the device (e.g.,

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

initialization, periodic self-tests, and calibration) to reduce spurious anomaly detections. The proposed anomaly detection approach can handle such time variations of periodic code via multiple strategies such as using different trained classifiers for different operating modes (e.g., a different classifier for power-on initialization stage) or using a single classifier which is trained using data that includes a sufficient variety of operating modes. An external whitelisting mechanism can also be used during specific times (e.g., device initialization) to discard spurious anomaly detections. Including training data from different operating modes/stages does not tend to introduce false positives/negatives since the one-class SVM essentially learns a representation of the set of "good" multi-thread measurement signals and inclusion of multiple operating modes/stages simply corresponds to a bigger set of such good patterns to be learned by the one-class SVM. Utilizing separate classifiers for different operating modes provides the advantage that anomalous patterns of operating modes can be detected (e.g., an attack that causes a device to enter calibration mode at an inappropriate time). Also, the proposed methodology applies even if the code on the CPS device is not periodic as long as the code has a sufficient repeatability of temporal patterns of code execution.

- Since HPCs characterize computations over a window of time, anomaly detection is addressed in the context of a sliding window of time and the entire time window is classified as anomalous or baseline. To reduce false positives and false negatives, the baseline vs. anomalous classifications over a **sequence of partially overlapping sliding windows** are utilized to generate the overall anomaly detection output. Since the time windows are partially overlapping and entire time windows are classified as baseline vs. anomalous, the majority voting does not introduce an opportunity for the attacker to intermittently introduce code modifications. Even an intermittent modification will be caught since the modification will appear in multiple of the partially overlapping time windows (the time shift between the time windows is much smaller than the widths of the time windows).
- HPCs are collected by an on-device **software** component in this paper. However, the HPCs can also be **monitored via hardware** debugging interfaces. Such remote monitoring by an external trusted device ensures that the monitor is safe from manipulation.
- Being a hardware-level functionality that is included in all modern processors, HPCs offer a **zero-hardware-cost** approach to anomaly detection. Furthermore, since the measurement mechanisms are based on hardware counters, collection of HPC readings is very lightweight in terms of computational overhead. HPC-based monitoring can be implemented even on tiny embedded devices without affecting real-time performance. In the Wago PLC utilized in experimental studies in this paper, the HPC measurement process was seen to utilize around 10% CPU and around 5 MB of memory. These CPU and memory requirements included writing of the measurements to a SQLite database; these requirements can be further reduced when required (e.g., by writing each reading directly to a text file or streaming to an output port) to fit even more constrained platforms. Also, it is to be noted that the actual anomaly detection is performed on a separate computational device (e.g., a portable single board computer or a laptop) and does not impose any CPU/memory overhead on the embedded device.
- Real-time detection of ***a priori*** **unknown malware/modifications** is addressed by monitoring anomalies relative to a baseline. A database of malware signatures is not needed. Anomaly detection uses a machine learning classifier over sliding windows of time. The approach balances robustness to stochastic variability under normal operating conditions (to reduce spurious anomaly detections) and sensitivity to anomalies due to code modifications (to reduce misses of anomaly detections).
- As seen in Section III-D, the proposed approach can **robustly detect anomalies** introduced by very small modifications of the real-time code on the embedded PLC. Many of the attacks considered in Section III-D involve changes of one or two lines of code. In this context, it is to be noted that a code modification involving changing, for example, a single parameter such as a control gain in the code might not be detectable from HPC readings. Instead, such a modification would have to be addressed by a complementary process-aware monitoring technique (e.g., [2]) in a multi-layered cyber-security approach in which the proposed approach would form one layer. However, it is to be noted that accomplishing any non-trivial objective of the attacker (such as overwriting sensor/actuator values, exfiltrating information, etc.) would, in general, require non-trivial code changes, which would be observable as changes in HPC patterns that would be detectable using the proposed approach. Making modifications while preserving all HPC counts is extremely difficult under real-world conditions and would require the adversary to have intricate familiarity with the hardware platform, code being monitored, and monitoring mechanism (e.g., HPCs being monitored, sampling rates, etc.). However, if a sufficiently motivated and able adversary were to craft a code modification that has no measurable HPC impact (or any other process-visible impact), then such a code modification would not be detectable.
- Real-time monitoring **scales to a variety of devices (32-bit and 64-bit Intel and ARM)** and can monitor specific or all processes. The approach is being tested on additional processor architectures. Guidelines for effective sets of feature types are being studied as well as comparison with methods such as neural networks for feature extraction and classification. While an autoencoder is an alternative approach to SVM-based anomaly detection, one advantage of a one-class SVM over an autoencoder is that an SVM learns an explicit representation of the distribution of the "good" examples in the training data while an autoencoder learns to reconstruct "good" signals, but not necessarily *not reconstruct* "bad" signals. Variants of autoencoder structures (e.g., variational autoencoder,

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

adversarially trained autoencoders) can possibly be used instead and will be addressed in future work. Additionally, an autoencoder (or a variational autoencoder) can potentially be used to extract features as part of the proposed approach (i.e., using the matrix of multi-threaded HPC readings over a time window as the input and target output of an autoencoder and using features from the low-dimensional hidden layer as part of the feature vector for the proposed approach).

- HPC measurements in a multiprocessing environment typically have stochastic variability. The proposed approach can detect anomalies relative to the stochastic characterization of the baseline operation. To reduce stochastic variability due to timing inaccuracies, HPCs can be **measured every some number of instructions rather than some amount of time**.

REFERENCES

[1] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, 2016.

[2] F. Khorrami, P. Krishnamurthy, and R. Karri, "Cybersecurity for control systems: A process aware perspective," *IEEE Design & Test Magazine*, vol. 33, no. 5, pp. 75–83, 2016.

[3] H. Salehghaffari, P. Krishnamurthy, and F. Khorrami, "A game theoretic approach to design a resilient controller for a nonlinear discrete system," in *Proceedings of the IFAC World Congress*, Toulouse, France, July 2017, pp. 387–392.

[4] E. Byres and J. Lowe, "The myths and facts behind cyber security risks for industrial control systems," in *Proceedings of the VDE Kongress*, vol. 116, 2004, pp. 213–218.

[5] N. Falliere, L. O. Murchu, and E. Chien, "W32. Stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, 2011.

[6] "ICS-CERT year in review – 2014," [Online]: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2014_Final.pdf, 2014.

[7] "NCCIC/ICS-CERT Year in Review – 2015," [Online]: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2015_Final_S508C.pdf, 2015.

[8] ICS-CERT, "ICS-CERT year in review," [Online]: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2016_Final_S508C.pdf, 2016.

[9] C. Blask, "ICS Cybersecurity: Water, water everywhere," [Online]: http://www.infosecisland.com/blogview/18281-ICS-Cybersecurity-Water-Water-Everywhere.html, Nov. 2011.

[10] J. Robertson and M. Riley, "Mysterious '08 Turkey pipeline blast opened new cyberwar," [Online]: http://www.bloomberg.com/news/articles/2014-12-10/mysterious-08-turkey-pipeline-blast-opened-new-cyberwar, Dec. 2014.

[11] R. J. Turk, "Cyber incidents involving control systems," [Online]: https://inldigitallibrary.inl.gov/sti/3480144.pdf, Oct. 2005.

[12] D. Kravets, "Feds: Hacker disabled offshore oil platforms' leak-detection system," [Online]: http://www.wired.com/2009/03/feds-hacker-dis/, Mar. 2009.

[13] D. Kushner, "The real story of Stuxnet," [Online]: http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet, Feb. 2013.

[14] E. Kovacs, "Cyberattack on German steel plant caused significant damage," [Online]: http://www.securityweek.com/cyberattack-german-steel-plant-causes-significant-damage-report, Dec. 2014.

[15] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proceedings of the ACM workshop on Scalable trusted computing*, 2011, pp. 71–76.

[16] P. Stewin, J.-P. Seifert, and C. Mulliner, "Poster: Towards detecting dma malware," in *Proceedings of ACM conference on Computer and communications security*, 2011, pp. 857–860.

[17] Y. Xia, Y. Liu, H. Chen, and B. Zang, "CFIMon: Detecting violation of control flow integrity using performance counters," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012, pp. 1–12.

[18] X. Wang and R. Karri, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *Proceedings of the ACM Design Automation Conference*, 2013.

[19] X. Wang, S. Chai, M. Isnardi, S. Lim, and R. Karri, "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," *Proceedings of the ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 1, p. 3, 2016.

[20] X. Wang and R. Karri, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 485–498, 2016.

[21] 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 *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 559–570.

[22] A. Tang, S. Sethumadhavan, and S. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, ser. Lecture Notes in Computer Science. Springer Verlag, 2014, vol. 8688, pp. 109–129, proceedings of the Springer International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2014, Gothenburg, Sweden.

[23] M. B. Bahador, M. Abadi, and A. Tajoddin, "HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *Proceedings of IEEE International Conference on Computer and Knowledge Engineering*, 2014, pp. 703–708.

[24] A. Garcia-Serrano, "Anomaly detection for malware identification using hardware performance counters," *arXiv preprint arXiv:1508.07482*, 2015.

[25] X. Wang, C. Konstantinou, M. Maniatakos, R. Karri, S. Lee, P. Robison, P. Stergiou, and S. Kim, "Malicious firmware detection with hardware performance counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 160–173, 2016.

[26] V. Jyothi, X. Wang, S. K. Addepalli, and R. Karri, "Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks," in *Proceedings of IEEE International Conference on VLSI Design and Embedded Systems*, 2016, pp. 587–588.

[27] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, "On the detection of kernel-level rootkits using hardware performance counters," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2017, pp. 483–493.

[28] M. F. B. Abbas, S. P. Kadiyala, A. Prakash, T. Srikanthan, and Y. L. Aung, "Hardware performance counters based runtime anomaly detection using SVM," in *Proceedings of the 2017 TRON Symposium*, Tokyo, Japan, Dec. 2017.

[29] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and A. Chattopadhyay, "RAPPER: Ransomware prevention via performance counters," *arXiv preprint arXiv:1802.03909*, 2018.

[30] N. Goldenberg and A. Wool, "Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 63–75, June 2013.

[31] Z. Zheng and A. L. N. Reddy, "Safeguarding building automation networks: The-driven anomaly detector based on traffic analysis," in *Proceedings of the International Conference on Computer Communication and Networks*, Vancouver, BC, Canada, Aug. 2017.

[32] "PAPI (Performance Application Programming Interface)," http://icl.utk.edu/papi.

[33] H. Amrouch, P. Krishnamurthy, N. Patel, J. Henkel, R. Karri, and F. Khorrami, "Emerging (un-)reliability based security threats and mitigations for embedded systems," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, Seoul, South Korea, Oct. 2017, pp. 1–10.

[34] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature Extraction: Foundations and Applications, Studies in Fuzziness and Soft Computing*. New York: Springer Verlag, 2006.

[35] C. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer Verlag, 2006.

[36] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond (adaptive computation and machine learning)*. Cambridge, MA: The MIT Press, 2001.

[37] V. Vapnik, *The nature of statistical learning theory*. New York: Springer Verlag, 1999.

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2923577, IEEE Transactions on Information Forensics and Security

[38] K. P. Murphy, "Machine learning: a probabilistic perspective," in *Adaptive Computation and Machine Learning series*. Cambridge, MA: The MIT Press, 2012.
[39] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer Verlag, 2009.
[40] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. New York: Cambridge University Press, 2009.
[41] J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & Chemical Engineering*, vol. 17, no. 3, pp. 245–255, 1993.
[42] A. Keliris, H. Salehghaffari, B. Cairl, P. Krishnamurthy, M. Maniatakos, and F. Khorrami, "Machine learning-based defense against process-aware attacks on industrial control systems," in *Proceedings of the 2016 International Test Conference*, Fort Worth, TX, Nov. 2016.
[43] P. Krishnamurthy, F. Khorrami, R. Karri, D. Paul-Pena, and H. Salehghaffari, "Process-aware covert channels using physical instrumentation in cyber-physical systems," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2761–2771, Nov. 2018.
[44] "Wago programmable fieldbus controllers," http://www.wago.us/products/components-for-automation/modular-io-system-ip-20-750753-series/plc/overview.
[45] "OpenPLC (open source PLC)," https://www.openplcproject.com/.
[46] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," in *IEEE Global Humanitarian Technology Conference*, San Jose, CA, Oct. 2014, pp. 585–589.

Prashanth Krishnamurthy received his B.Tech. degree in electrical engineering from Indian Institute of Technology, Chennai in 1999, and M.S. and Ph.D. degrees in electrical engineering from Polytechnic University (now NYU), Brooklyn, NY in 2002 and 2006, respectively. He is currently a Research Scientist and Adjunct Faculty with the Department of Electrical and Computer Engineering at NYU Tandon School of Engineering, NY, and a Senior Researcher with FarCo Technologies, NY. He has co-authored over 120 journal and conference papers in the broad areas of autonomous systems, robotics, and control systems. He has also co-authored the book "Modeling and Adaptive Nonlinear Control of Electric Motors" published by Springer Verlag in 2003. His research interests include autonomous vehicles and robotic systems, multi-agent systems, sensor data fusion, robust and adaptive nonlinear control, resilient control, path planning and obstacle avoidance, machine learning, real-time embedded systems, electromechanical systems modeling and control, cyber-physical systems and cyber-security, decentralized and large-scale systems, high-fidelity and hardware-in-the-loop simulation, and real-time software implementations.

Ramesh Karri is a Professor of ECE at New York University. He co-directs the NYU Center for Cyber Security (http://cyber.nyu.edu). He also leads the Cyber Security thrust of the NY State Center for Advanced Telecommunications Technologies at NYU. He co-founded the Trust-Hub (http://trust-hub.org) and organizes the Embedded Systems Challenge (https://csaw.engineering.nyu.edu/esc), the annual red team blue team event. Ramesh Karri has a Ph.D. in Computer Science and Engineering, from the UC San Diego and a B.E in ECE from Andhra University. His research and education activities in hardware cybersecurity include trustworthy ICs; processors and cyber-physical systems; security-aware computer-aided design, test, verification, validation, and reliability; nano meets security; hardware security competitions, benchmarks and metrics; biochip security; additive manufacturing security. He has published over 240 articles in leading journals and conference proceedings. Karri's work on hardware cybersecurity received best paper nominations (ICCD 2015 and DFTS 2015) and awards (ACM TODAES 2018, ITC 2014, CCS 2013, DFTS 2013 and VLSI Design 2012). He received the Humboldt Fellowship and the National Science Foundation CAREER Award. He serves on the editorial boards of several IEEE and ACM Transactions (TIFS, TCAD, TODAES, ESL, D&T, JETC). He served as an IEEE Computer Society Distinguished Visitor (2013-2015). He served on the Executive Committee of the IEEE/ACM Design Automation Conference leading the SecurityDAC initiative (2014-2017). He delivers invited keynotes, talks, and tutorials on Hardware Security and Trust Trust (ESRF, DAC, DATE, VTS, ITC, ICCD, NATW, LATW, CROSSING, etc.). He co-founded the IEEE/ACM NANOARCH Symposium and served as program/general chair of conferences (IEEE ICCD, IEEE HOST, IEEE DFTS, NANOARCH, RFIDSEC and WISEC). He serves on several program committees (DAC, ICCAD, HOST, ITC, VTS, ETS, ICCD, DTIS, WIFS).

Farshad Khorrami received his Bachelors degrees in Mathematics and Electrical Engineering in 1982 and 1984 respectively from The Ohio State University. He also received his Master's degree in Mathematics and Ph.D. in Electrical Engineering in 1984 and 1988 from The Ohio State University. Dr. Khorrami is currently a professor of Electrical & Computer Engineering Department at NYU where he joined as an assistant professor in Sept. 1988. His research interests include adaptive and nonlinear controls, robotics and automation, unmanned vehicles, cyber security for cyber-physical systems, embedded systems security, machine learning, and large-scale systems and decentralized control. Prof. Khorrami has published more than 260 refereed journal and conference papers in these areas. His book "Modeling and Adaptive Nonlinear Control of Electric Motors" was published by Springer Verlag in 2003. He also has thirteen U.S. patents on novel smart micro-positioners and actuators, control systems, security, and wireless sensors and actuators. He has developed and directed the Control/Robotics Research Laboratory at Polytechnic University (Now NYU). His research has been supported by the ARO, NSF, ONR, DARPA, ARL, AFRL, NASA, and several corporations. Prof. Khorrami has served as general chair and conference organizing committee member of several international conferences. Prof. Khorrami has also commercialized UAVs as well as development of auto-pilots for various unmanned vehicles.

1556-6013 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.