Stealthy Rootkits in Smart Grid Controllers

Prashanth Krishnamurthy, Hossein Salehghaffari, Shiva Duraisamy, Ramesh Karri, and Farshad Khorrami
Control/Robotics Research Laboratory
NYU Center for Cybersecurity
Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, NY, 11201 USA
(prashanth.krishnamurthy,h.saleh,sd3165,rkarri,khorrami)@nyu.edu

***Abstract*—This paper presents a stealthy and persistent attack on a Cyber-Physical System (CPS), namely the smart grid and a multi-layer approach to detect such an attack. The attack on the CPS controller uses a rootkit-based malware. When activated, the rootkit overwrites operator commands to the smart grid relays while evading detection by the operator control station. The rootkit sends valid replies to the operator while corrupting the controller operation through a dynamically loaded library, which is hidden by the rootkit. The attack persists even when the controller stops and restarts since the rootkit automatically restarts the process with the malicious library by using a background daemon, which the rootkit hides from user-space tools. Using a high-fidelity simulation of the smart grid CPS, we show that the attack drastically impacts the CPS, especially when the adversary strategically chooses the target relays to attack. We design an ensemble of detectors to detect the attack and uncover its persistence and insertion mechanisms. The detector uses measures such as hardware performance counters (HPCs), change detection in binary signatures, change detection in system calls, and detection of hidden processes and file system entries.**

***Index Terms*—Anomaly Detection, Cyber Security, Rootkit, Stealthy Attacks, Actuator Spoofing, Programmable Logic Controller, Malware, Power Grid Resiliency, Resilient Control.**

I. Introduction

The interplay of cyber and physical components in a Cyber-Physical System (CPS) allows an attack to impact the physical processes by exploiting cyber vulnerabilities [1]–[4]. Embedded controllers (e.g., Programmable Logic Controllers – PLCs) are common in CPS such as the smart grid. While distribution, maintenance, and reconfiguration benefit from networking and remote programmability, they also allow remote attacks on the cyber and physical parts of the CPS [5]–[15]. An attacker can use compute, communication, and network vulnerabilities to insert malicious code. Supply chain vulnerabilities are relevant where the malicious code is "designed" into the controller, remaining latent until triggered. An attack may 1) modify the controller logic to impact stability, performance, or safety of the physical process, 2) spoof information from sensors, spoof outputs of actuators, or communication between operator control stations and the controllers, and 3) ex-filtrate sensitive data, add back doors, or give access to unauthorized users. Preventing and detecting attacks requires a multi-layered approach [4], [16]–[18] from a myriad of vantage points: network, on-device, and process-aware monitors.

In this paper, we study a realistic, stealthy rootkit enabled malware in embedded controllers and associated detection methodologies.The considered attack seeks to undermine stability, efficiency, and safety of the smart grid. The malware injects a dynamically loaded library into the controller software to overwrite commands from an operator to a relay in the grid. Using open-source kernel-space and user-space rootkit libraries, a daemon that replaces the OpenPLC controller process with a modified process (that has a malicious library hooked to it) is hidden from user-space tools. This malicious library hooks into the process to override crucial I/O routines in the controller code. Further, it sends modified operator commands to the physical I/O and incorrect status messages to the operator control station. This attack is validated in a Hardware-In-The-Loop (HITL) testbed for the smart grid [19], [20]. To detect the malicious modifications to the controller and uncover the malware persistence and insertion mechanisms, we develop a multi-pronged approach to anomaly detection that combines run-time integrity verification using hardware performance counters (HPCs), detectors to enumerate binary objects of the process and kernel-level system call entries to detect changes from a baseline, and detectors to detect anomalies in the virtual (`/proc` in Linux) and physical file systems to detect hidden processes and file system entries.

The rootkit-based stealthy attack and its HITL-based testing is described in Section III. The anomaly detection techniques, which use multiple complementary techniques, are described in Section IV. Concluding remarks are provided in Section V.

II. Related Work

CPS and ICS security is increasingly relevant [4], [16], [18] due to many reported attacks [5]–[15]. There has been substantial work on stealthy rootkits [21]–[25]. A process-aware attack in [26] implements a PLC rootkit that modifies the control commands sent to the power grid testbed. A stealthy attack in [27] modifies the control logic on the PLC while hidden from software at the control center. [28] is a rootkit detector based on phase-space analysis of power

This work was supported in part by the U.S. Office of Naval Research under Award N00014-18-1-2672 and by DARPA under AFRL contract FA8750-16-C-0179. The authors are associated with NYU Control/Robotics Research Laboratory (CRRL) and Ramesh Karri is associated with NYU CCS and NYU-AD CCS-AD.

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00012

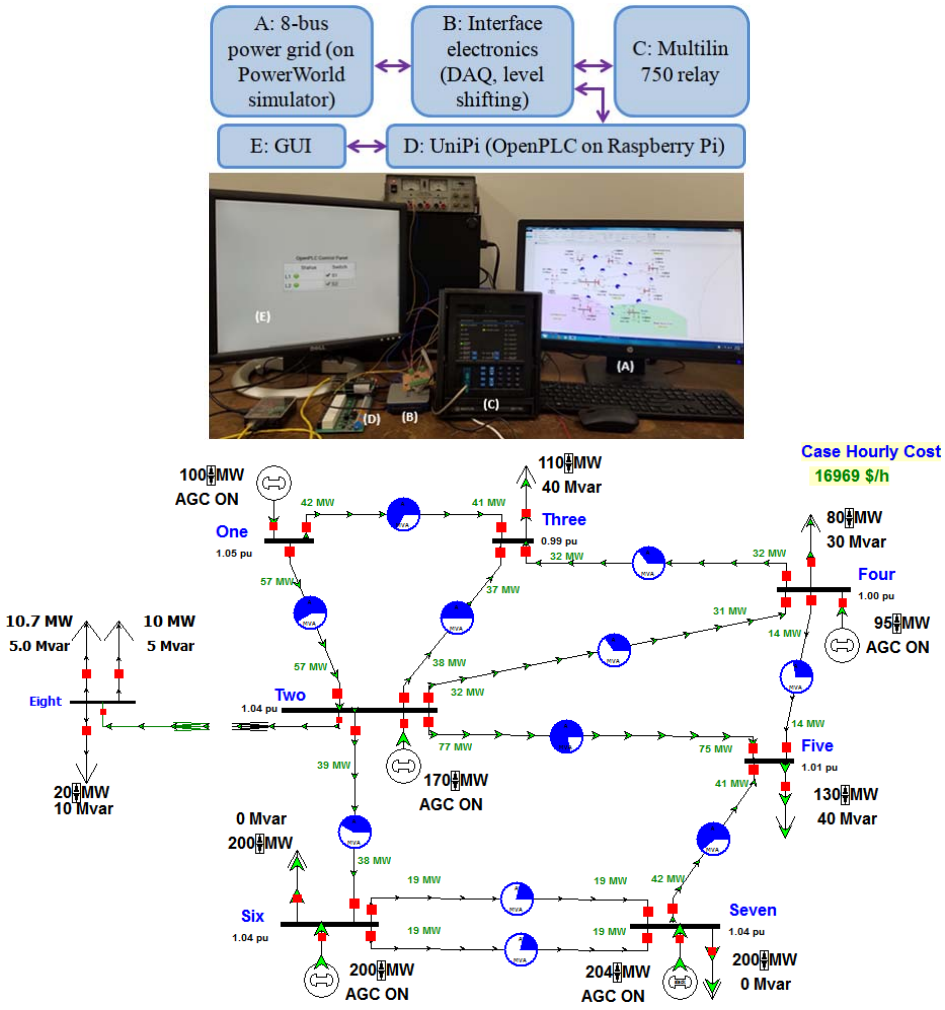

Fig. 1. The HITL smart grid testbed comprises of: (A) a power grid simulator for an 8-bus power grid (shown in the bottom picture), (B) a data acquisition (DAQ) board with level shifting, (c) GE Multilin 750 relay, (D) UniPi SCADA PLC emulator (Raspberry Pi microcontroller that controls the OpenPLC), (E) GUI operator terminal (that monitors relay status) on another Raspberry Pi, with which the UniPI communicates with the GUI via Modbus protocol.

voltage measurements. These attacks have spurred interest in monitoring and anomaly detection in CPS. Approaches that monitor network traffic have been proposed [29], [30]. While [29] monitors Modbus/TCP traffic for abnormalities in messages using a deterministic finite automaton model of normal traffic, [30] monitors BACnet traffic in building automation systems. A variety of rootkits and rootkit detection methods have been addressed in [31]–[40]. HPCs have been leveraged to detect anomalies in code execution [41]–[55]. HPCs have been used in a Virtual Machine environment to detect kernel control-flow modifying rootkits [44], [46]. HPCs provide a lightweight hardware-level hook to monitor applications running on a processor.

HITL simulators have been developed to assess security in the smart grid CPS [56]–[60]. In [57], the testbed integrates a real-time power simulator with LabView as a SCADA system and PXI modules as intelligent electronic devices (IEDs). A CPS testbed in [59] studies attacks on the power grid. The HITL testbed has a simulated power grid, sensors, controllers, and communication network. Another HITL testbed with an SEL 351S protection and OPAL-RT simulator is used in [58].

III. A Stealthy Attack on the Smart Grid CPS

A. Attack Objectives

In the CPS controller attack, an adversary modifies a process that runs the control logic on an embedded controller. This control logic is specified via a Structured Text program or a graphical ladder logic and is loaded onto the PLC as an executable. In one instantiation of the attack, the adversary gains unauthorized access to the PLC (e.g., using a vulnerability in the network protocol used to program the PLC or via unauthorized access) and implants the malware.

B. Attack Outline

- *Step 1:* When an adversary gains physical/network access to the device (or its components during manufacturing, integration, repair), he/she implants a malware on it. The malware has a malicious library intended to be pre-loaded into the PLC controller process, a daemon script, and a rootkit. The daemon watches for restarts of the controller process and relaunches the process with the malicious library. The rootkit masks existence of the malicious library and daemon by masking their file system entries and the running daemon process from user-space tools.
- *Step 2:* The malware is activated (e.g., immediately or by a time- or event-based trigger) during device operation.
- *Step 3:* Once activated, the malware overrides operator commands to relays in the system while sending appropriate status messages to the operator station. The operator believes that the relays are receiving the commands and changing their state.

While the attack spoofs actuator commands, the technique can be used to: alter control behavior (e.g., spoofing sensor readings, modify variables/logic in the controller), exfiltrate sensitive data from the device [61], and modify messages to feed erroneous information to HMI.

C. HITL Smart Grid CPS Testbed

We implement the attack on an OpenPLC embedded controller in a HITL testbed (in Figure 1) emulating the operation of an 8-bus power grid in the Power World software interfaced to a physical relay. The HITL testbed is along the lines of [60], [62], [63] (we would like to acknowledge the help of the authors of [60], [62], [63] in setting up the testbed). The connection of the relay to the grid is controlled through UniPi, a Raspberry Pi single board computer with an additional board that provides several Input/Output (I/O) interfaces for the PLC (including analog and digital I/O pins). A simple "operator terminal" graphical user interface (GUI) is implemented on a second Raspberry Pi using the pvbrowser process visualization browser software [64]. Via this GUI, the operator can control the open/closed status of the relay. The operator terminal and the UniPi are connected via Ethernet and communicate using the Modbus protocol. The command signal of the OpenPLC passes through the data acquisition board to change the status of the relay within the PowerWorld simulator. In real-life, the GE Multilin 750 relay is used for the primary protection and control for feeders on solidly grounded, high impedance grounded, or resonant grounded systems. The Multilin 750 has analog input/output interfaces for monitoring as well as RS232, RS485/RS422, and Ethernet ports for communications. EnerVista Software provides a tool to monitor information measured by the relay into a SCADA system.

D. Identifying Victim/Vulnerable Relays

After identifying ideal relays to target, an adversary can launch the attack in Section III-A to maximize damage. The prerequisite goal is to identify the critical relays in the network. By attacking these relays, he/she can disconnect the loads or the transmission lines from the network. This can cause congestion in the network and impact the transient stability and efficiency.The power network is characterized by the graph $G(V, E)$ where $V = \{1, 2, \ldots, N\}$ is the set of nodes, E is the set of transmission lines, and $N_i = \{j \in V | (i, j) \in E\}$ is the neighbor set of the i^{th} node. The most critical relays of transmission lines in the network can be identified using network theory [65], [66] as follows. The active power between nodes i, j in a lossless network is

$$P_{ij} = \frac{\theta_i - \theta_j}{X_{ij}} \tag{1}$$

where P_{ij} is the active power transmitted from node i to node j, θ_i is the phase angle of node i, and X_{ij} is the reactance of the transmission line connecting nodes i and j. The power flow through any transmission line is inversely proportional to the reactance of that line. The reactance can be considered as the weight of the edge connecting any two vertices in the graph $G(V, E)$ representing the power network. The attacker objective is to determine the subset of lines that when disrupted impact the network the most. This objective can be viewed in terms of determining the subset of lines with the most contribution to power flow distribution in the network. To determine this subset of lines, we use the "line betweenness index" as a metric [67]–[69]. The betweenness index of a transmission line is the total number of shortest paths that pass through the particular line in the network. The contribution of a transmission line to power flow distribution in a network is quantified in terms of number of times that the transmission line appears in the shortest paths between any two nodes in the network. Consequently, transmission lines with high betweenness index are beneficial for an attacker.

E. Effects of the Attack on the Smart Grid

The power network in Figure 1 is considered for HITL simulation in this study. This 8-bus network has 5 generators connected to buses 1, 2, 4, 6, and 7. It has 10 loads connected to buses 1, 3, 4, 5, 6, 7, and 8. The power generation and power consumption of each generator and load are specified in Figure 1. Table I shows the betweenness index and reactance of the lines of the grid. From the results in the table, the transmission line connecting the nodes 2 and 3 is critical. Eliminating this line from the network and re-computing the line betweenness yields the next critical line between nodes 4 and 5. These two lines are good attack targets. The efficiency of the network G is:

$$J = \frac{1}{N(N-1)} \sum_{i \neq j \in G} x_{ij} \tag{2}$$

where N is the number of nodes and x_{ij} is the shortest electrical distance between nodes i and j. Figure 2 shows the efficiency of the power network for random and targeted attacks on the relays of the transmission lines.

Figure 3 shows the effects of attacking relays of the transmission lines on congestion in the power network. During the

TABLE I
BETWEENNESS INDEX AND LINE REACTANCE OF THE NETWORK.

Line No.	Betweenness	Reactance	Line No.	Betweenness	Reactance
L_{13}	7	4.16	L_{45}	7	4.16
L_{12}	0	16.67	L_{57}	2	25
L_{23}	12	5.55	L_{26}	10	16.67
L_{24}	0	25	L_{28}	7	12.5
L_{25}	10	8.33	L_{28}	4.16	5
L_{34}	0	33.33			

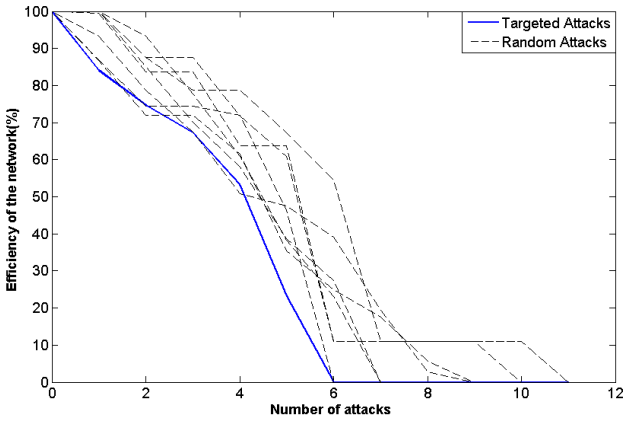

Fig. 2. Efficiency of the network for targeted and random attacks on relays on the transmission lines.

attack (opening the relay on a critical line), the power flow through the line connecting buses 1 and 3 reaches 84% of the maximum tolerable power by the line and the power flow through the line connecting buses 2 and 5 reaches 87% of the maximum tolerable power flow. Congestion in lines increases maintenance cost of the lines and the nodal price of the buses corresponding to the congested lines. Figure 4 shows impacts of the attacks on transient response of the bus voltages. The voltage level at different buses in normal condition should be between 1.05 pu-0.95 pu (pu denotes "per unit"). Although a steep change is seen in the voltage level of the buses when the transmission lines disconnect from the network, all bus voltages in Figure 4 remain in this range. Figure 5 shows the frequency response. Transient stability simulations do not show regulation of frequency back to nominal (60 Hz). In addition to (or in conjunction with) attacking relays on transmission lines, the attacker can also target load relays to degrade the performance and stability of the CPS.

F. Overwriting Operator Command by a Malicious Dynamically Loaded Library

Overriding operator commands to the relays connected to the hardware I/O pins on the OpenPLC can be done either at the point where the operator commands are read into the OpenPLC or where the commands are written out to the hardware I/O pins. We adopt the former. The OpenPLC process uses the `read` and `write` library functions in the libc library to read from a file descriptor to read commands sent by the operator and to write back status messages to the operator terminal. These functions are used with software on Linux to access any file (or I/O). Overriding these functions can change the behavior of software without recompiling the software and without accessing the source code.

To implement an attack that overrides the `read` and `write` functions, we wrote a C code with two functions with identical signature to the system `read` and `write` functions. We compiled the C code as a dynamically loadable shared object file. When the target process (the OpenPLC controller) starts with this dynamically loaded library specified in the LD_PRELOAD environment variable, the library is pre-loaded into the process. Hence, the `read` and `write` functions implemented in this library override the system's libc implementations. In the first invocations of the `read` and `write` functions in the C code for this library, the addresses of the real system `read` and `write` functions are looked up using `dlsym` and cached. In subsequent invocations, cached pointers to the system `read` and `write` functions are used to redirect calls to the real system functions when desired.

In the `read` function in the C code for the dynamically loaded library, the real `read` function is first used to read from the file descriptor. It checks if the file system descriptor (provided as a function argument) matches the specific file descriptor that is being targeted. If it does, the command

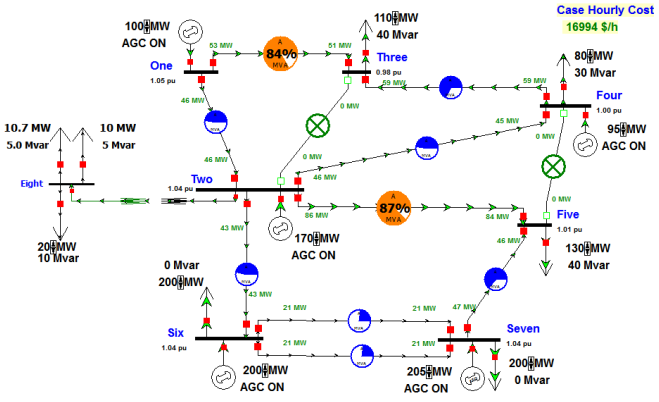

Fig. 3. Congestion in the network when relays on L_{23} and L_{45} open at $t = 1$ s and $t = 2$ s.

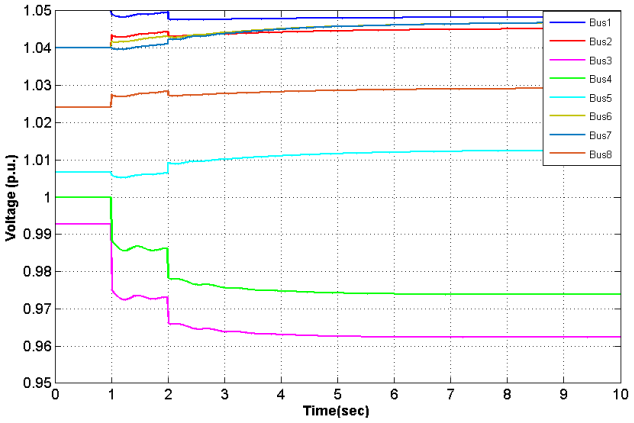

Fig. 4. Voltage on the buses when relays on L_{23}, L_{45} open at $t = 1$ s and $t = 2$ s.

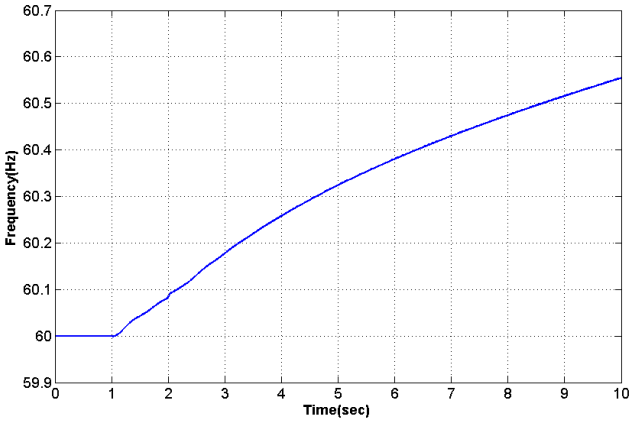

Fig. 5. Frequency of the network when the relays on L_{23} and L_{45} open at $t = 1$ s and $t = 2$ s.

(an array of bytes) sent by the operator is parsed to detect if the command contains state changes for the specific I/O pins that are being targeted by the malware. If it does, the commands sent by the operator for these I/O pins are stored while these entries in the command are replaced by malware-specified values. In the `write` function, it checks if the file system descriptor matches the specific file descriptor that is being targeted. If it does, the array of bytes being written is parsed to check whether this array contains status messages for the specific I/O pins being targeted. If it does, then the appropriate entries in the array of bytes are modified so as to contain the *expected* status messages corresponding to the previous command received using the `read` function from the operator terminal. This modified array of bytes is written by calling the real `write` function. Thus, when the operator sends a command to a relay connected to the OpenPLC, the malware intercepts it and sends malicious commands to the relay. When sending back status messages from the relay, the malware reports modified status messages to make it appear as if the actuator commands by the operator are processed. Apart from modifying the commands and messages, the malware goes through the same steps as the normal code (i.e., writes to the I/O pins and reads status updates from the pins). Hence, the timing properties (e.g., delay between the operator sending a command and getting back confirmation via updated status messages) are identical to the original code making the device appear normal to an operator monitoring the GUI.

G. Rootkit-Based Stealthy Malware Insertion

To insert the malicious library that overrides the `read` and `write` functions described above into the OpenPLC controller process, a daemon script was implemented. This daemon runs in the background and polls (at fixed time intervals) the process listing on the device to check if an unmodified controller process (i.e., without the malicious library pre-loaded) is running. If it detects an unmodified controller process, it kills it and immediately relaunches a replacement process with the malicious library pre-loaded. The malware thus has multiple artifacts on the device including the daemon script and the malicious library on the file system, and the running daemon process. To elude detection, a rootkit masks their existence from user-space tools. Two rootkits are considered (one kernel-based and the second in user-space), which can be used separately or in combination. The kernel-based rootkit is implemented using the Diamorphine open-source library [36] and the LD_PRELOAD-based rootkit is implemented using the bedevil open-source library [37]:

- The Diamorphine rootkit [36] loads a kernel module which overwrites specific entries in the Linux system call table. The entries for system calls for reading file system entries (`getdents` and `getdents64`) and for sending signals to a process (`kill`) are overwritten. By overriding the system calls for reading file system entries, file system entries (e.g., based on a prefix in the name) can be made invisible to user-space tools. Also, entries in the `/proc` virtual file system can be hidden and thus, the daemon script process can be hidden. Diamorphine hides the kernel module by removing the entry from the list maintained by the kernel to track loaded kernel modules.
- The bedevil LD_PRELOAD-based rootkit [37] operates in user space, but has similar capabilities of hiding processes and file system entries as a kernel-based rootkit. Bedevil inserts a pre-loaded library into every new process. This library masks system libc functions in a manner similar to the masking of the `read` and `write` functions in Section III-F. For example, when a user-space tool attempts to list file system entries (including in `/proc`) using any function in libc, it inadvertently calls the replacement function provided by the pre-loaded dynamic library. This malicious function filters the list of file system entries. This filtering can be done based on user/group IDs, process names, etc. *Contents* of files in the virtual (e.g., `/proc`) and physical file system are also filtered when read by user-space tools. The listing

of loaded mapped files of a process is filtered to hide the existence of the pre-loaded library in the process.

While the rootkits hide the artifacts of the malware (the running daemon, its file system entries, and the malicious library) and also hide themselves, we will build anomaly detectors to detect such malware.

IV. Methodologies to Detect Stealthy Malware

An ensemble of anomaly detectors can detect the malware. While some of the anomaly detectors are baseline-dependent (i.e., detect anomalies relative to a baseline), the other detectors are baseline-independent (i.e., do not require measurements from a known-good device). HPCs (Section IV-A) detect changes in run-time characteristics of code execution. Listings of mapped memory regions of a process (Section IV-B) detect unexpected dynamically loaded libraries or changes in libraries. Reading the system call table (Section IV-C) detects changes in memory addresses in the system call table or in the system call handler functions. Techniques to detect anomalies in process listings and file system entries (Section IV-D) uncover processes and file system entries hidden by a rootkit. These methods detect the stealthy malware from Section III by detecting anomalous code execution and uncovering the malware artifacts.

A. HPCs

HPCs count various types of events (e.g., instructions, branches, etc.) during code execution. A time series of HPC measurements offers a temporal profile of the code being executed. While modern processors typically support a large number of types of events that can be counted using HPCs, the number of HPCs that can be measured simultaneously is typically constrained. The quad-core ARM Cortex on the Raspberry Pi allows simultaneous reading of 6 HPCs. HPCs for numbers of instructions, branches, stores, cycles, L1 instruction cache misses, and L2 data cache misses are used in this study and HPCs were read at a sampling rate of 1 kHz. The HPCs are collected per-thread for each of the 3 threads in the OpenPLC controller process using a separate measurer process. The HPC measurer uses the PAPI (Performance Application Programming Interface) library [70] and connects remotely to the target process and reads time series of HPCs corresponding to each of the threads in the target process. The HPC measurer is implemented in C++ and is compiled on a separate computer into a statically linked executable and deployed to the OpenPLC Raspberry Pi via ssh.

A baseline data set of HPC readings over a time interval of 120 s was collected from a good device. Half of this data set was used to train a one-class Support Vector Machine (SVM) classifier [71]. A sliding time window of length 0.25 s was considered (with a time shift of 0.01 s between successive time windows). One of the 3 threads in the OpenPLC controller process was quiescent. The mean and standard deviations (over the time window) of the measured HPCs for the 2 non-quiescent threads was used to construct the feature vector for the one-class SVM classification of normal/outlier. A data set of length 120 s was collected after deploying the malware.

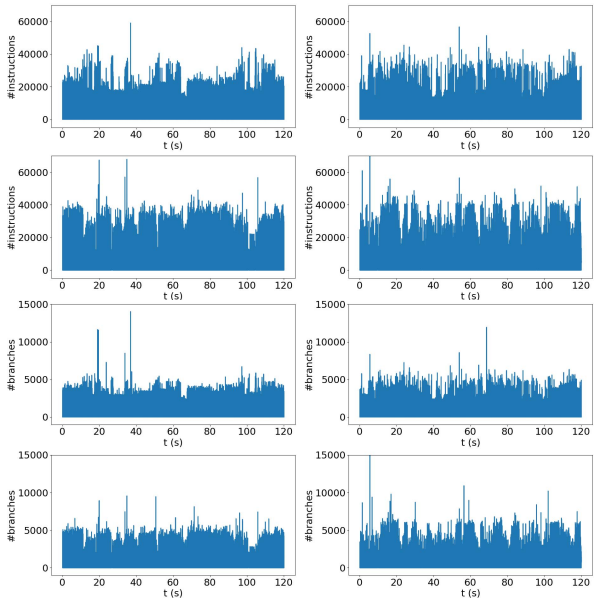

Fig. 6. HPC measurements for instructions and branches for each of the 2 non-quiescent threads in the OpenPLC controller under normal (left column) and malware-infested (right column) conditions.

The time series of HPC measurements from the good device and the malware-infested device are in Figure 6. While these time series look somewhat similar from a macroscopic view (except for intermittent non-deterministic spikes), the one-class SVM trained on baseline data accurately distinguishes between data from the good and malware-infected device. The one-class SVM is trained only on data from the baseline device. The SVM was tested on the second half of the baseline data set and the data set collected under malware. A sliding window of 40 normal/outlier detection is used (with majority voting) to output normal/outlier labels. The classification accuracy for baseline data is 90.06% (i.e., 10% false positives where normal data was marked as anomalous) and the classification accuracy for data from the malware-infested device was 93.94% (6.0% of false negatives –where anomalous data was marked normal). HPC-based detector indicates something is anomalous on the device.

B. Binary Object Enumeration and Change Detection

The mapped memory regions of a running process with a specific process ID (PID) can be read from the `/proc/PID/maps` file of the virtual `/proc` file system on Linux. The list of mapped memory regions includes the path names of mapped files (which includes the executable and dynamically loaded libraries). The list of mapped files and the signatures of these files (e.g., SHA1 hashes) can be used to detect if any unexpected libraries have been dynamically loaded and to detect if any libraries have been modified relative to a baseline. By reading the `/proc/PID/maps` file for the

OpenPLC controller, this comparison relative to a baseline is a simple change detector that can flag the malicious library loaded via LD_PRELOAD in Section III-F.

A rootkit can elude this detection to some extent. While the Diamorphine rootkit does not alter the read contents of the `/proc/PID/maps` file, the bedevil rootkit filters the `/proc/PID/maps` file. However, the LD_PRELOAD mechanism to dynamically pre-load a library does not affect statically linked executables. Hence, by cross-compiling similar to the HPC measurer described in Section IV-A to generate a statically linked executable from a separate computer, the true contents of the `/proc/PID/maps` file can be read. Using this on the bedevil rootkit, we can detect the malicious library in Section III-F and the bedevil-inserted library as unexpected dynamically loaded libraries in the OpenPLC controller.

C. Detection of Anomalous System Calls

Overwriting of system call (syscall) entries and changes to syscall code can be detected by reading appropriate parts of the kernel memory. This can be accomplished from user space using the `/dev/kmem` file or by inserting a Linux kernel module to enumerate memory addresses in the syscall table and contents of those addresses. The `/dev/kmem` file is a device file that provides direct access to the kernel virtual memory. While `/dev/kmem` is not enabled by default on newer versions of the Linux kernel, a significant number of real-world embedded devices often run older versions of the kernel and provide access to `/dev/kmem`. On devices on which this file is not available, a kernel module can be loaded instead. On the Raspbian operating system distribution on the Raspberry Pi, this file is not available. Hence, a kernel module was developed instead to read syscall entries from kernel memory. With either the `/dev/kmem` file or from a kernel module, the syscall table location can be identified using the symbol table at `/proc/kallsyms` or the `System.map` file for the running kernel or by scanning the kernel memory for a known syscall address such as `sys_close`. While newer kernel versions disable some of these methods for obtaining syscall table memory location for security, embedded devices often do provide these. On Raspberry Pi, the `/proc/kallsyms` file is available and provides the syscall table address in the `sys_call_table` entry.

Once the syscall table is located, memory contents starting from this location provide addresses of the syscall routines. Starting from each of these memory addresses, reading some number of bytes provides a signature of the contents of that syscall handler function. With a kernel module, syscall function addresses and memory contents of the initial segments of the syscall functions are exfiltrated to user space using the kernel log. Alternatively, a special device file is created and written to from the kernel module to exfiltrate the information to user space. Comparing the exfiltrated information of a device under test with readings from a known-good device enables detecting changes relative to a baseline. A simple memory address based outlier detection provides a baseline-independent anomaly marker that is effective on embedded devices with a small number of kernel modules. When rootkits such as Diamorphine deploy and insert new kernel modules to overwrite syscall table entries, the memory addresses of these syscall function replacements are in vastly different memory address locations (numerically) and can be detected by clustering and outlier analysis. The outlier-based anomaly detection is baseline-independent. Using these techniques, Diamorphine is reliably detected. The bedevil-based rootkit does not insert a kernel module and does not overwrite syscall entries.

D. Detection of Hidden Processes and File System Entries

The basic strategy to detect hidden processes is to find mismatches between different approaches to list running processes including: Using a command such as `ps`; Listing numeric entries in the `/proc` directory; Brute-force scanning of PIDs by attempting to change directories into each possible directory of form `/proc/PID`. With Diamorphine, process lists from the first two methods match with each other while the process list from the third method includes hidden processes. With bedevil, a statically linked executable lists the full contents of `/proc`; hence, process lists from the second and third methods match while the process list from the first method lacks entries corresponding to hidden processes. Since a mismatch could be due to a process starting/exiting between listings using different methods, process lists are acquired twice with each method for consistency. Once a hidden process is detected, its attributes can be retrieved from its `/proc/PID` directory (e.g., executable from the `/proc/PID/exe` link, current working directory from `/proc/PID/cwd`, command line from `/proc/PID/cmdline`). Similar to hidden process detection, hidden file system entries can be detected in a baseline-independent fashion by checking for mismatches when using different methods to obtain file system directory counts including: Using `ls` to list directory contents and counting number of sub-directories; Listing directory contents using `readdir` and counting number of sub-directories; Using the `lstat` system call to read number of hard links to a directory. Mismatches between the results flag hidden sub-directories. With Diamorphine, numbers from the first two methods match while the number from the third method is higher when there are hidden entries (since it includes hard links from hidden sub-directories). With bedevil, a statically linked executable yields matching numbers from the second and third method (corresponding to contents including hidden directories) while the first method misses hidden directories.

V. Conclusions

The novel aspects of the study include: (1) A stealthy malware that endures controller restarts and transparently overrides operator commands while remaining undetectable from the operator terminal. (2) Demonstrating the malware on a HITL smart grid CPS shows the real-world applicability of the malware and the impacts that such malware can have on stability, efficiency, and safety of the CPS. (3) Synergistic detection methodologies that uncover such stealthy malware.

REFERENCES

[1] A. A. Cárdenas, S. Amin, and S. Sastry, "Research challenges for the security of control systems," in *Proceedings of the USENIX workshop on Hot Topics in Security*, July 2008.

[2] A. Cárdenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry, "Challenges for securing cyber physical systems," in *Proceedings of the Workshop on Future directions in cyber-physical systems security*, 2009, p. 5.

[3] A. S. Uluagac, V. Subramanian, and R. Beyah, "Sensory channel threats to cyber physical systems: A wake-up call," in *Proceedings of the IEEE Conference on Communications and Network Security*, 2014, pp. 301–309.

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

[5] 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.

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

[7] "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.

[8] "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.

[9] 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.

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

[11] 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.

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

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

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

[15] 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.

[16] 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.

[17] 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 IEEE International Test Conference*, Fort Worth, TX, Nov. 2016, pp. 1–10.

[18] 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.

[19] "OpenPLC (open source PLC)," https://www.openplcproject.com/.

[20] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," in *Proceedings of the IEEE Global Humanitarian Technology Conference*, San Jose, CA, Oct. 2014, pp. 585–589.

[21] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction," in *Proceedings of the ACM Conference on Computer and communications security*, Oct. 2007, pp. 128–138.

[22] D. Lobo, P. Watters, X.-W. Wu, and L. Sun, "Windows rootkits: Attacks and countermeasures," in *Proceedings of the Cybercrime and Trustworthy Computing Workshop*, July 2010, pp. 69–78.

[23] T. Arnold and T. A. Yang, "Rootkit attacks and protection: a case study of teaching network security," *Journal of Computing Sciences in Colleges*, vol. 26, no. 5, pp. 122–129, May 2011.

[24] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitoring through vmm-based "out-of-the-box" semantic view reconstruction," *ACM Transactions on Information and System Security*, vol. 13, no. 2, p. 12, Feb. 2010.

[25] H. Sayadi, H. M. Makrani, S. M. P. Dinakarrao, T. Mohsenin, A. Sasan, S. Rafatirad, and H. Homayoun, "2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, Feb. 2019, pp. 728–733.

[26] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! attacking plcs with physical model aware rootkit." in *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2017.

[27] S. Kalle, N. Ameen, H. Yoo, and I. Ahmed, "Clik on plcs! attacking control logic with decompilation and virtual plc," in *Proceedings of the Binary Analysis Research Workshop, Network and Distributed System Security Symposium*, Feb. 2019.

[28] J. A. Dawson, J. T. McDonald, J. Shropshire, T. R. Andel, P. Luckett, and L. Hively, "Rootkit detection through phase-space analysis of power voltage measurements," in *Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE)*, Oct. 2017, pp. 19–27.

[29] 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.

[30] 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.

[31] A. Bunten, "Unix and linux based rootkits techniques and countermeasures," in *Proceedings of the Forum of Incident Response and Security Teams Conference*, Budapest, Hungary, June 2004.

[32] A. Todd, J. Benson, G. Peterson, T. Franz, M. Stevens, and R. Raines, "Analysis of tools for detecting rootkits and hidden processes," in *Proceedings of the IFIP International Conference on Digital Forensics*, P. Craiger and S. Shenoi, Eds., Orlando, FL, 2007, pp. 89–105.

[33] D. R. Wampler, "Methods for detecting kernel rootkits," 2007, ph.D. dissertation, University of Louisville, Kentucky.

[34] M. Leibowitz, "Horse pill: A new kind of linux rootkit," in *Black Hat USA*, Las Vegas, NV, July 2016.

[35] S. Eresheim, R. Luh, and S. Schrittwieser, "The evolution of process hiding techniques in malware - current threats and possible countermeasures," *Journal of Information Processin*, vol. 25, pp. 866–874, Sep. 2017.

[36] "Diamorphine Linux kernel module rootkit," https://github.com/m0nad/Diamorphine.

[37] "bedevil Linux LD_PRELOAD rootkit," https://github.com/naworkcaj/bdvl.

[38] "rkhunter rootkit hunter," http://rkhunter.sourceforge.net.

[39] "unhide forensic tool," http://www.unhide-forensics.info/?Linux.

[40] "chkrootkit tool to check for signs of a rootkit," http://www.chkrootkit.org.

[41] 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.

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

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

[44] 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.

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

[46] 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.

[47] 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 International Symposium on Computer Architecture*, 2013, pp. 559–570.

[48] A. Tang, S. Sethumadhavan, and S. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, ser. Springer Verlag Lecture Notes in Computer Science, 2014, vol. 8688, pp. 109–129.

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

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

[51] 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.

[52] 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 the IEEE International Conference on VLSI Design and Embedded Systems*, 2016, pp. 587–588.

[53] 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.

[54] 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.

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

[56] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye, and D. Nicol, "Scada cyber security testbed development," in *Proceedings of the North American Power Symposium*, Sep. 2006, pp. 483–488.

[57] R. Liu, C. Vellaithurai, S. S. Biswas, T. T. Gamage, and A. K. Srivastava, "Analyzing the cyber-physical impact of cyber events on the power grid," *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2444–2453, Sep. 2015.

[58] S. Poudel, Z. Ni, and N. Malla, "Real-time cyber physical system testbed for power system security and control," *International Journal of Electrical Power & Energy Systems*, vol. 90, pp. 124–133, Sep. 2017.

[59] B. Chen, N. Pattanaik, A. Goulart, K. L. Butler-Purry, and D. Kundur, "Implementing attacks for modbus/tcp protocol in a real-time cyber physical system test bed," in *Proceedings of the IEEE International Workshop Technical Committee on Communications Quality and Reliability*, Toronto, Canada, May 2015, pp. 1–6.

[60] A. S. Musleh, C. Konstantinou, M. Sazos, A. Keliris, A. Al-Durra, and M. Maniatakos, "Gps spoofing effect on phase angle monitoring and control in an RTDS based hardware-in-the-loop environment,," *IET Cyber-Physical Systems: Theory and Applications*, vol. 2, pp. 180–187, June 2017.

[61] P. Krishnamurthy, F. Khorrami, R. Karri, D. Paul-Pena, and H. Salehghaffari, "Process-aware covert channels using physical instrumentation in cyber-physical systems," *IEEE Transaction on Information Forensics and Security*, vol. 13, no. 11, pp. 2761–2771, Nov. 2018.

[62] C. Konstantinou, E. Chielle, and M. Maniatakos, "Phylax: Snapshot-based profiling of real-time embedded devices via jtag interface," in *Proceedings of the Design, Automation and Test in Europe Conference Exhibition*, Dresden, Germany, March 2018, pp. 869–872.

[63] A. Keliris, C. Konstantinou, and M. Maniatakos, "Ge Multilin SR protective relays passcode vulnerability," in *Black Hat USA*, Las Vegas, NV, July 2017.

[64] "pvbrowser HMI and SCADA," https://github.com/pvbrowser/pvb.

[65] E. Bompard, D. Wu, and F. Xue, "Structural vulnerability of power systems: A topological approach," *Elsevier Electric Power Systems Research*, vol. 81, no. 7, pp. 1334–1340, 2011.

[66] X. Chen, K. Sun, Y. Cao, and S. Wang, "Identification of vulnerable lines in power grid based on complex network theory," in *Proceedings of the IEEE Power Engineering Society General Meeting*, June 2007, pp. 1–6.

[67] S. Jin, Z. Huang, Y. Chen, D. Chavarría-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing*, 2010, pp. 1–7.

[68] E. Bompard, D. Wu, and F. Xue, "The concept of betweenness in the analysis of power grid vulnerability," in *IEEE Complexity in Engineering*, 2010, pp. 52–54.

[69] H. Bai and S. Miao, "Hybrid flow betweenness approach for identification of vulnerable line in power system," *IET Generation, Transmission and Distribution*, vol. 9, no. 12, pp. 1324–1331, 2015.

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

[71] P. Krishnamurthy, R. Karri, and F. Khorrami, "Anomaly detection in real-time multi-threaded processes using hardware performance counters," *IEEE Transaction on Information Forensics and Security*, 2019, early access on-line version, June 2019.