Malicious Firmware Detection with Hardware Performance Counters

Xueyang Wang, *Student Member, IEEE,* Charalambos Konstantinou, *Student Member, IEEE,* Michail Maniatakos, *Member, IEEE,* Ramesh Karri, *Member, IEEE,* Serena Lee, Patricia Robison, Paul Stergiou, and Steve Kim

Abstract—Critical infrastructure components nowadays use microprocessor-based embedded control systems. It is often infeasible, however, to employ the same level of security measures used in general purpose computing systems, due to the stringent performance and resource constraints of embedded control systems. Furthermore, as software sits atop and relies on the firmware for proper operation, software-level techniques cannot detect malicious behavior of the firmware. In this work, we propose ConFirm, a low-cost technique to detect malicious modifications in the firmware of embedded control systems by measuring the number of low-level hardware events that occur during the execution of the firmware. In order to count these events, ConFirm leverages the Hardware Performance Counters (HPCs), which readily exist in many embedded processors. We propose a comparison-based technique to detect malicious modifications in firmwares with simple control-flows. For firmwares with more complex control-flows, we use machine learning techniques to automatically extract the relations among different hardware events. This method significantly reduces the number of pre-stored valid HPC signatures without compromising the detection accuracy. Finally, we reduce the consumption of local resources by implementing a remote-based detection mechanism. We evaluate the detection capability and performance overhead of the proposed technique on various types of firmware running on ARM- and PowerPC-based embedded processors. Experimental results demonstrate its practicality and effectiveness.

Index Terms—Firmware, hardware performance counters, attacks, detection.

1 Introduction

The interconnection of embedded devices as well as their intelligent capabilities, make embedded systems an integral part in a wide range of applications. Such applications include mobile phones, measurement equipment, home automation technologies, and even mission critical industrial control systems. For example, power grid transformation into a more dynamic and interactive smart system is highly dependent on embedded devices [1]. According to a recent study, the embedded systems market is expected to generate over $2 trillion in revenue during 2015 [2]. Furthermore, taking into account the Internet-of-Things (IoT) continuous advancements, embedded devices as IoT nodes will constitute the front lines where decisions are made [3].

The data and programs written onto the non-volatile memory of a device, make the system functional as they connect hardware and software modules. All these necessary instructions consist the firmware layer of a system. As a result, firmware has become a major focus for targeted attacks due to its substantial role in an embedded system operation. Effective security mechanisms are, therefore, of paramount importance in order to provide protection against firmware malicious attacks.

The exploitation of firmware vulnerabilities in order to attack embedded devices has been reported for various types of embedded systems. For example, Miller demonstrated how to reprogram a smart battery through firmware modifications [4]. It has also been shown that arbitrary malware can be injected into printers due to vulnerabilities of the remote firmware update procedure [5]. A recent large-scale analysis of firmware images revealed 38 previously unknown vulnerabilities in over 693 firmware images [6]. Many other firmware modification attacks exist in a wide range of devices such as hard-drives [7], routers [8], [9], [10], [11], keyboards [12] and mice [13], multi-function peripherals [14] and PBX equipments [15].

Firmware attacks have also been demonstrated in embedded systems used for mission critical applications. For instance, Checkoway *et al.* have presented an attack on the electronic control units of a car due to a custom firmware upload [16]. Hanna *et al.* found that automated external defibrillators used for treating cardiac arrhythmias would accept counterfeit firmware updates [17]. In addition, a proof-of-concept experiment has been used to demonstrate how a modified version of firmware can be updated and uploaded to a Programmable Logic Controller (PLC) [18].

Providing security countermeasures to firmware malicious actions should be the initial step for protecting not only embedded systems but any kind of computing system. If the underlying firmware is not trusted then any other mechanism implemented at the Operating System (OS) or application level cannot be trusted. To date, several tech-

- *X. Wang, C. Konstantinou and R. Karri are with the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, Brooklyn, NY 11201, USA. E-mail: {xw338, ckonstantinou, rkarri}@nyu.edu*
- *M. Maniatakos is with the Department of Electrical and Computer Engineering, New York University Abu Dhabi, Abu Dhabi, 129188, UAE. E-mail: michail.maniatakos@nyu.edu*
- *S. Lee, P. Robison, P. Stergiou and S. Kim are with Consolidated Edison, New York, NY 10003, USA. E-mail: {leese, robisonp, stergioup, kims}@coned.com*

niques have been proposed and implemented to detect malicious firmwares. However, these techniques require either extra hardware components (e.g. Trusted Platform Module) or code verification software designs. These designs often rely on sufficient resources for which an embedded system has limited capabilities (e.g. computation resources, power consumption, memory usage, communication bandwidth, etc.).

In this work, we propose a low-cost technique to detect firmware modifications on embedded systems, overcoming the challenges and constraints of existing schemes. The proposed technique, called ConFirm, is directly encapsulated in embedded systems in order to indicate whether or not a firmware has been maliciously modified. The detection mechanism is based on the monitoring of low-level hardware events: since a program is composed of a sequence of various types of instructions, the program characteristics can be uniquely captured by the total occurrences of hardware events during its execution, as well as the relation between the counts of different monitored events. Examples of hardware events include branches, retired instructions, returns. The execution of a maliciously modified firmware would result in different occurrences of monitored hardware events compared to the benign firmware code.

Access to low-level hardware events for running processes can be achieved by using Hardware Performance Counters (HPCs). Based on the architecture of the embedded device processor (e.g. ARM, MIPS, PowerPC etc.) and its Performance Monitoring Unit (PMU), HPCs measure numerous and different types of hardware events [19]. While HPCs are mainly used for performance tuning, ConFirm leverages HPCs for detecting malicious firmware modifications. Since the monitored events are automatically counted by HPCs at the hardware level, ConFirm does not have any real-time computing constraints. In addition, it does not require extra hardware. Therefore, the resource cost and overhead of the monitored embedded system remain significantly low. Summarizing, our contributions are as follows:

- We propose and design ConFirm, a host-based firmware validation tool taking into account the limited resources of embedded systems. ConFirm leverages existing hardware features (HPCs) to identify whether or not the firmware of an embedded system is malicious.
- A prototype of ConFirm is implemented on ARM- and PowerPC-based embedded platforms. Modification attacks are applied to firmware samples of commercial devices in order to demonstrate the feasibility of the technique. We also evaluate the performance and storage overhead on the monitored system.
- We use machine learning techniques to automatically extract the relations among different hardware events. This method significantly reduces the number of pre-stored valid HPC signatures without compromising the detection accuracy. Finally, we reduce the consumption of local resources by implementing a remote-based detection mechanism.

The paper is organized as follows: Section 2 discusses the prerequisites of using ConFirm. Section 3 presents the overview of ConFirm as well as implementation details. The evaluation results are shown in Section 4. Section 5 describes the detection with support vector machine based machine learning. The remote-based detection technique is presented in Section 6. Related work on firmware attacks and detection mechanisms are given in Section 7. Finally, we conclude the paper in Section 8.

2 PREREQUISITES

In this section, we introduce the adversarial model used throughout the study, as well as the essential requirements for deploying ConFirm on an embedded device.

2.1 Threat Model

We focus on attacks which modify the firmware of embedded systems. Specifically, we target attacks that modify the firmware code in a way that introduces execution of malicious code or circumvent firmware critical functions in the code flow execution. An attacker has access to exploits that allow execution of arbitrary malicious code. For instance, the attacker can perform attacks based on code injection and code reuse attacks such as the return-to-libc or return oriented attacks. In addition, these exploits can either be zero-day firmware vulnerabilities or firmware code modifications. The malicious firmware alternation can be injected to the system online, i.e. during a firmware update, or offline, i.e. by uploading the malicious image to the device (requiring system reboot).

2.2 Hardware Performance Counters

HPCs are special-purpose registers built into the PMU of a modern microprocessor in order to store information about hardware events [20], [21]. Event selectors specify the user-defined choice of hardware events to monitor. Since HPCs were originally designed for performance debugging of complex software, software developers heavily rely on HPC-based profilers to understand the runtime behavior of a program and tune its performance.

HPCs provide access to detailed performance information with lower overhead and higher accuracy than software profilers. Furthermore, HPCs do not require any source code modifications. The number of available HPCs, as well as the number of hardware events, vary from one processor model to another [22]. For example, while the ARM V8 Cortex-A53 core has 4 HPCs, and can count 62 events [23], the fourth generation of 32-bit PowerPC microprocessors (such as the PowerPC 7450) incorporates only 35 events and 4 HPCs [24].

2.2.1 Computational Path Analysis with HPCs

A subroutine of the firmware generally has multiple computational paths in its control-flow graph. Each computational path goes through different code blocks and therefore executes different code. The executed code generates different vectors in terms of the counts of hardware events. A computational path is primarily determined by the input applied to the subroutine and the state of data structures used by the subroutine. Let $C(E_x)$ denote the count of event x from the execution of a targeted computational path. If m hardware

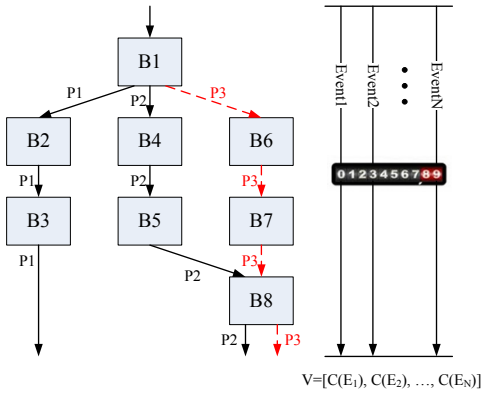


Fig. 1: Computational path analysis with HPCs. The execution of the valid paths P1 and P2 in a monitored subroutine generates different vectors V_1 and V_2 in terms of the occurrences of low-level hardware events. As an example, malicious execution could go through path P3, generating an unexpected HPC vector V_3 which is different from V_1 and V_2.

events are monitored simultaneously, an HPC vector V with m elements can be obtained as follows:

$$V = [C(E_1), C(E_2), C(E_3), ..., C(E_m)] \quad (1)$$

With a selected set of hardware events, a vector can be consistent for every execution as discussed in detail in Section 3.4 (except the situation that the path contains a loop with a dynamic number of iterations, where a measurement only targeting the loop is required). As a result, each vector can be considered as the signature of a specific path. ConFirm uses such an HPC signature to verify the execution of computational paths in a subroutine of the monitored firmware. The computational path analysis is presented in Figure 1.

Control-flow modifications underlie a wide range of common attacks. In other words, an attacker typically hijacks the original control-flow of the victim program in order to perform any malicious actions [25]. This is achieved by executing code or calling functions not included in the control-flow. Examples include stack-based buffer-overflow and heap-based "jump-to-libc" attacks [26], [27]. Enforcing control-flow integrity (CFI) as a defense can thwart such attacks. However, CFI techniques are expensive with overheads that may be as high as 25% [28] to 50% [29]. In addition, CFI techniques are unable to perform instrumentation of modules separately [30]. They typically require all modules of a system (libraries included) to be available at the instrumentation time.

2.3 Root of Trust

The booting of an embedded system is a multi-phase process. Each phase is responsible for loading the next phase. The first code that is executed in this *boot sequence* resides in a part of memory that is protected from accidental erasure or corruption and is typically called boot Read-Only-Memory (ROM). The boot ROM code locates and executes the second software phase, called the pre-loader. The pre-loader initializes memory and an appropriate subset of the peripheral devices in order to access and load the next phase (bootloader) into main memory. The bootloader is responsible for locating and loading the OS or firmware which will run the embedded system applications on the device. If the embedded device is a non-OS based system, then the firmware tasks are executed in an infinite loop (super loop based execution). Because the three phases (boot ROM, pre-loader and bootloader) reside in a reserved read-only area of the system, they constitute the root of trust of the embedded system [31].

Since write-protected memories are secure and protected by design, we leverage the root of trust by incorporating ConFirm HPC profiling-based scheme within the bootloader. As a result, ConFirm remains in a trusted area, immune to malicious modifications

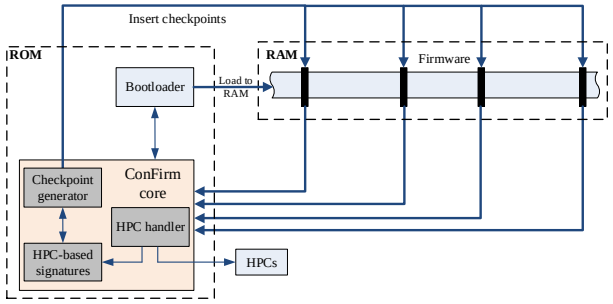


Fig. 2: High-level structure of ConFirm. The ConFirm core consists of three components: an insertion module that inserts checkpoints to the monitored firmware, an HPC handler that drives the HPCs and a database that stores valid HPC-based signatures.

3 ConFirm Overview

The high-level structure of ConFirm is shown in Figure 2. A legacy bootloader is extended with the ConFirm core. The core consists of three components: a) an insertion module that places checkpoints to the monitored firmware, b) an HPC handler that drives the HPCs, and c) a database that stores valid HPC-based signatures. All these components are stored in write-protected non-volatile memory. This prevents attacks from compromising ConFirm while still allowing authorized updates[1].

The advantages of ConFirm can be summarized as follows:

- ConFirm core resides in the bootloader, thus is difficult to be detected or disabled by an adversary.
- ConFirm monitoring can be instrumented within any executable, and does not depend on the functionality of the monitored firmware.
- ConFirm directly utilizes the hardware features of the host platform, bypassing the overhead associated with the software layers.

1. Similar to general-purpose computers using BIOS for booting, an embedded system bootloader can only be updated outside the nominal operation of the device.

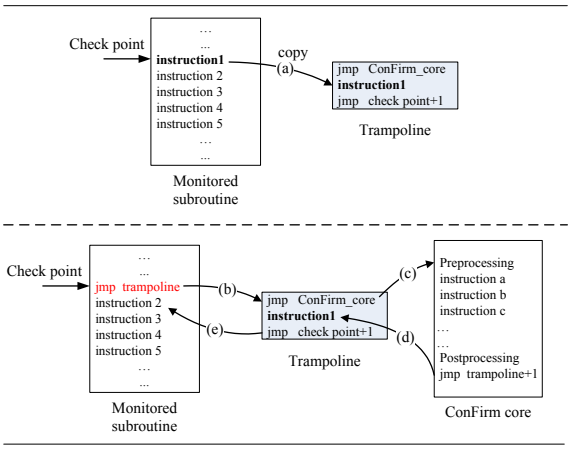


Fig. 3: Control-flow interception in ConFirm. A Detour style inline hooking mechanism is employed to redirect the control-flow to the ConFirm when the execution of the firmware reaches a checkpoint.

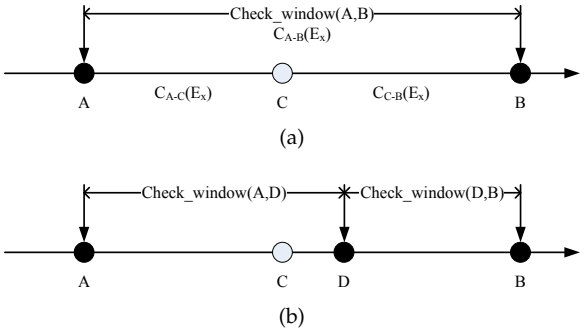


Fig. 4: Checkpoint randomization to avoid prediction of check windows.

3.1 ConFirm Modules

The bootloader transfers the control to ConFirm after the firmware image is loaded into memory and before its execution. The checkpoint generator of ConFirm automatically inserts checkpoints to pre-determined locations across the firmware. The firmware validation starts when the first checkpoint is reached. A pair of two adjacent checkpoints forms a check window during which the execution of the firmware is monitored. Each checkpoint is the end point of the previous check window and the starting point of the next check window. Hence, the check windows are allocated contiguously without any gaps following the control-flow of the firmware. More details about checkpoint insertion are presented in Section 3.2.

After checkpoint insertion, the control-flow proceeds to the HPC handler. The HPC handler executes a small sequence of instructions in order to configure, initialize and enable the HPCs. With the HPCs enabled, the control is transferred back to the bootloader which initiates the execution of the firmware.

Once the execution reaches a checkpoint, ConFirm intercepts the control-flow and redirects it to the core module. The core then communicates with the HPC handler and the HPC-based signature database. Specifically, the event counts for the previous check window are read and compared with the corresponding signatures in the database. Then the HPCs are reset for the next check window and the execution of the monitored firmware continues. The HPCs keep counting the occurrences of the hardware events until the next checkpoint is reached.

3.2 Checkpoint Insertion

The prototype implementation of ConFirm uses a Detours style inline hooking mechanism for control-flow interception and checkpoint insertion [32]. The checkpoint insertion technique is shown in Figure 3. In the monitored firmware, the instruction at a checkpoint is first copied and preserved in a certain memory location in RAM which is called a ”trampoline” (arrow (a) in Figure 3). Then, the firmware instruction is replaced with a jump instruction to transfer the control to the trampoline (b). In the trampoline, another jump instruction is executed that targets a location in the ConFirm core module (c). The sequence of instructions in the Confirm core module ends with a jump back to the trampoline after the check is completed (d). The copied instruction which is replaced in the original subroutine, is executed in the trampoline before the control is transferred back to the monitored subroutine (e). The next instruction to be executed in the subroutine is the one right after the replaced instruction. With such an inline hooking, the functionality of the subroutine remains the same after applying the check. For each checkpoint, a dedicated trampoline needs to be allocated since the instruction copied and preserved for each checkpoint is different. Also, the jump instructions in different trampolines target different locations in the ConFirm core module.

3.2.1 Checkpoint Randomization

Checkpoints are inserted at the entry and exit points of each monitored subroutine. This opens up an opportunity for an attacker to manipulate the HPCs, evading the detection. Figure 4 (a) shows an example in which an attacker can bypass the check if he knows the checkpoint locations. Points A and B are the entry and exit points of a computation path $P(A, B)$ of a monitored subroutine. Thus, two checkpoints are inserted at locations A and B. Location C is an intermediate point of the computation path $P(A, B)$.

Assume that the attacker modifies the control-flow between A and C. The count of event x from A to C after the modification is $C_{A-C}(E_x)$. If the attacker knows the “good” value of $C_{A-B}(E_x)$ (the count of the check window (A, B)), he can modify the control-flow between C and B to make $C_{A-C}(E_x) + C_{C-B}(E_x)$ equal to $C_{A-B}(E_x)$, and thus bypassing the check. To prevent attackers from predicting the check windows, checkpoints are also inserted at arbitrary locations in the monitored subroutines (besides the entry and exit points), as shown in Figure 4 (b). Such locations can be changed every time the system reboots or when the control is transferred at runtime to ConFirm. Specifically, the insertion module maintains multiple sets of locations where the checkpoints can be inserted. When the system boots one set of locations is selected randomly and checkpoints are inserted. Besides the boot-time checkpoint insertion, the insertion module also selects a new set

of locations and updates the checkpoints at runtime. The addresses of trampolines are also updated when a new set of checkpoints is inserted.

3.3 Comparison-based Detection

ConFirm uses a comparison-based two-phase detection as shown in Figure 5. In the offline profiling phase, the HPC signatures of all monitored execution paths are generated from a clean copy of the targeted firmware. In the online checking phase, the same monitored paths are measured and the runtime signatures are compared against the corresponding golden ones.

3.3.1 Offline Profiling

This phase is performed before the device is deployed. The clean copy of the firmware is executed on the device with checkpoints inserted at pre-determined locations, splitting the execution flow into multiple check windows. During each check window, the occurrences of m monitored hardware events for a computational path are measured and the HPC vector $V = [C(E_1), C(E_2), C(E_3), ..., C(E_m)]$ is obtained. The measurement is performed for all the n possible paths in the check window to complete the HPC signature of a check window $W = [V_1, V_2, V_3, ..., V_n]$. After total x check windows have been profiled, the HPC-based signatures of the monitored firmware $S = [W_1, W_2, W_3, ..., W_x]$ are generated and stored in the database as the golden reference signatures.

3.3.2 Online Checking

When a check is invoked during the online checking phase, ConFirm reads all the configured counters and obtains the runtime HPC vector V for the current check window. Then, ConFirm refers to the signature database and retrieves the reference vectors generated offline for the same check window. The runtime HPC vector V is matched with the offline references. If a match is found, indicating the tested vector V is valid, the control returns to the monitored firmware and execution resumes. In case the runtime vector does not match to any of the offline references, a deviation will be reported by ConFirm and appropriate actions, specified by the integrator, will be initiated. Possible reactions to anomaly detection include rebooting the system, generating an alarm and disabling the device.

3.4 HPC Selection

The hardware events on a platform can be categorized into different groups, such as load/store events, cache events and execution unit events. This gives us many options to model an HPC signature. However, not all the available hardware events are good candidates for modeling a repeatable signature, as the occurrence of some events depends on the current, unpredictable system state. For such events, the occurrences vary dramatically given different executions. Extracted signatures must be repeatable and robust. Experimental results presented in Section 4.1 elaborate on the selection of proper HPCs for the target firmwares.

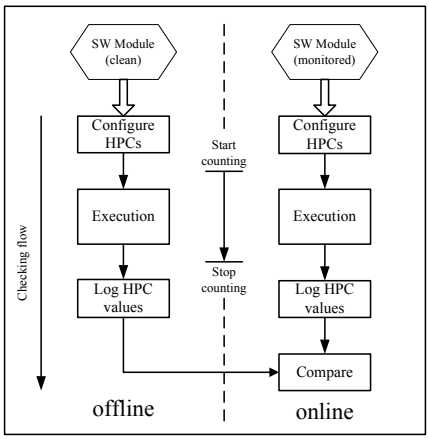


Fig. 5: Offline profiling phase (left-hand side) and online checking phase (right-hand side) of ConFirm comparison-based two-phase detection.

TABLE 1: The hardware events of ARM Cortex A15 (a) and PowerPC e300c3 (b) with small C.V against system disturbances. A smaller C.V indicates a better repeatability of an HPC-based signature.

(a)

Hardware event	C.V (%)
BRANCH instruction executed	0.72
INSTRUCTION architecturally executed	0.93
RETURN instruction speculatively executed	1.07
STORE instruction speculatively executed	1.27
LOAD instruction speculatively executed	1.27
Average over all tested events (∼70)	**18.9**

(b)

Hardware event	C.V (%)
BRANCH instruction completed	1.05
Completed INSTRUCTION	1.13
LOAD micro-ops completed	1.59
STORE micro-ops completed	1.78
BRANCH instruction MISPREDICTED	2.35
Average over all tested events (∼40)	**16.7**

4 Experimental Results

In this section we demonstrate the detection capability of ConFirm. We also evaluate the performance and storage overhead when ConFirm is enabled.

4.1 ConFirm Capability

To demonstrate the effectiveness of ConFirm, we test our technique with two real-world firmwares of embedded systems on two different platforms: Samsung Exynos Arndale [33] and Freescale MPC8308RDB [34]. The Arndale platform has an ARM Cortex-A15 processor with 6 HPCs and can support 70 different hardware events [35]. The MPC8308RDB platform contains a PowerPC e300c3 core which has 4 HPCs and can support 40 available hardware events [36].

In order to determine which hardware events are more robust for signature modeling, we execute a set of software

TABLE 2: ConFirm detection capability. The numbers are the event count deviations $D(E_x)(P_{test}, P_{ref\_x})$ (%) of the malicious path from the valid paths in different check windows of the monitored subroutines from three ARM-based firmware ((a), (b), (c)) and three PowerPC-based firmware ((d), (e), (f)). For each path, the bold number indicates the largest deviation among all events. The tested path (malicious) is not matched to any valid path indicating a successful detection.

(a)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	27.3	33.3	37.5	**50.0**
2	77.1	71.4	**81.5**	50.0
3	73.3	71.4	**76.2**	60.0
4	51.5	**60.0**	58.3	0.0
5	65.9	50.0	**75.0**	60.0
6	69.8	71.4	**76.2**	33.3
7	62.8	**71.4**	66.7	33.3
Check window 2				
1	77.8	33.3	**150.0**	0.0
2	44.8	60.0	16.7	**66.7**
Check window 3				
1	33.8	**175.0**	31.6	40.0
2	**16.5**	15.8	8.7	12.0

(b)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	39.9	41.6	50.0	**76.2**
2	29.1	35.9	57.7	**84.2**
3	35.6	28.9	71.4	**88.9**
4	30.8	47.4	33.3	**69.6**
Check window 2				
1	36.6	50.9	**80.0**	73.3
2	32.8	**87.5**	53.3	64.7
3	38.9	63.6	34.8	**84.6**
Check window 3				
1	20.7	22.9	46.7	**69.2**
2	19.4	21.6	53.8	**75.0**

(c)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	28.6	35.4	51.7	**52.7**
2	26.9	32.6	**59.7**	50.0
3	27.2	38.7	**72.1**	61.7
4	30.3	42.7	58.5	**67.4**
5	36.1	47.6	64.6	**85.3**
Check window 2				
1	31.0	46.9	**80.0**	79.2
2	24.4	53.6	**77.4**	67.9
3	32.0	71.4	82.8	**90.5**
Check window 3				
1	22.4	60.0	**76.0**	63.2
2	22.8	50.0	**73.1**	70.6

(d)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	65.0	**266.7**	78.6	250.0
2	10.0	10.0	0.0	**55.6**
3	41.4	**83.3**	16.7	33.3
4	6.5	22.2	4.2	**47.4**
5	5.7	10.0	7.4	**33.3**
Check window 2				
1	**95.8**	76.8	62.1	30.0
2	19.5	51.1	**70.6**	30.0
3	**65.0**	46.7	37.5	44.0
Check window 3				
1	30.3	12.0	16.7	**47.4**

(e)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	39.2	**61.5**	41.7	25.0
2	53.6	**78.4**	39.2	40.0
3	41.1	**56.3**	46.5	28.6
4	44.4	**64.5**	40.8	30.8
5	46.2	**72.7**	52.6	33.3
6	49.6	**75.5**	57.1	38.1
Check window 2				
1	45.0	**63.6**	60.0	33.3
2	45.5	68.3	**71.4**	40.0
3	56.9	**77.8**	55.6	50.0
4	53.4	**87.5**	65.2	46.2
Check window 3				
1	48.7	**90.5**	57.9	33.3

(f)

Path	Hardware event (E_x)			
	I	B	L	S
Check window 1				
1	22.1	7.7	**25.0**	21.2
2	23.3	10.8	**25.9**	22.9
3	24.7	11.1	**27.5**	21.6
4	26.3	12.3	**32.6**	25.6
5	28.0	14.0	**32.6**	31.4
Check window 2				
1	24.4	6.5	21.1	**30.4**
2	**26.0**	7.3	22.9	25.0
3	**29.4**	9.1	25.8	29.2
4	**32.6**	9.7	30.8	24.1
Check window 3				
1	21.3	9.5	22.2	**23.1**
2	23.5	13.3	**26.7**	25.0

modules multiple times on the same platform. The repeatability is quantified with the Coefficient of Variation (C.V). Table 1 lists the hardware events with the smallest C.V for ARM Cortex A15 and PowerPC e300c3. These events are the most robust, thus are considered as good candidates for ConFirm.

In this experiment, we perform proof-of-concept malicious modifications to a monitored subroutine for each firmware. We then measure the event count deviations of the malicious path from all the valid paths in the monitored subroutines. Let $C_{P_{test}}(E_x)$ denotes the count of event E_x from the execution of a path under test P_{test}. Similarly, $C_{P_{ref\_y}}(E_x)$ denotes the event counts E_x from the execution of a valid reference path P_{ref_y}. The deviation of P_{test} from P_{ref_y} on event E_x is presented as follows:

$$D(E_x)(P_{test}, P_{ref\_y}) = \left| \frac{C_{P_{test}}(E_x) - C_{P_{ref\_y}}(E_x)}{C_{P_{ref\_y}}(E_x)} \right| \quad (2)$$

Considering the system disturbances, it is required to set a noise threshold N. In case $D(E_x)(P_{test}, P_{ref\_y})$ is greater than N, ConFirm suggests a malicious modification. Assume x events are monitored concurrently, where $1 \leq x \leq m$. If for all these events $D(E_x)(P_{test}, P_{ref\_y})$ is less than N, P_{test} is matched to the valid path P_{ref_y}. If no match is found in the checking procedure, a malicious modification will be reported by ConFirm. The results of the deviations $D(E_x)(P_{test}, P_{ref\_y})$ as well as the found threshold N are presented below for two commercial embedded systems[2].

4.1.1 Case Study 1: ARM-based Firmware

We first test ConFirm with a commercial ARM-based firmware. The embedded device is an all-in-one wireless access point and access gateway designed for use by public hot-spot providers and enterprises. The firmware contains a VxWorks real-time operating system [37].

2. We do not include the names of the target devices due to a nondisclosure agreement.

In this experiment a Denial of Service (DoS) attack is performed [38]. The attack targets the task scheduling module of the firmware. Specifically, we add a function hook to the *checkTaskSwitch* subroutine to modify the normal control-flow of the task scheduling algorithm. When a task with a specific ID is running it will occupy the processor without being switched out, thus will impact the availability of other tasks.

The modified firmware is evaluated and tested with ConFirm on the Samsung Exynos Arndale board. The bootloader used on the platform is U-Boot, a multi-platform, open-source, universal bootloader with comprehensive support for loading and managing boot images [39]. Owing to their repeatability as shown in Table 1, we choose 4 events to monitor: instruction architecturally executed (I), branch instruction executed (B), load instruction speculatively executed (L) and store instruction speculatively executed (S). The event RETURN instruction speculatively executed is not monitored since it does not occur in the monitored subroutine.

There are seven valid paths in the original *checkTaskSwitch* subroutine, named as P_{ref_1} to P_{ref_7}. At runtime the HPC-based signature will be compared with a subset of the valid signatures (those included in the randomly selected check window) to check if a match is found. The results are shown in Table 2 (a) for 3 randomly selected check windows.

The minimum deviation $D(E_x)$ for a malicious path P_{test} to be detected is 8.7% in check window 3 (among the events whose occurrences have changed). In this case, the monitored LOAD event defines the minimum noise detection threshold N. For example, a detection threshold of 5% is adequate to identify the malicious modifications in every chosen check window. Since there is no match between the tested malicious path and any valid path, the test case indicates a successful detection.

The results of similar experiments on another two ARM-based firmwares are presented in Table 2 (b) and (c).

4.1.2 Case Study 2: PowerPC-based Firmware

This firmware runs on a PowerPC-based microprocessor. The embedded system in this case study protects the power grid by tripping and reclosing distribution power lines. Specifically, the controller causes a recloser (i.e. circuit breaker) to trip and reclose in case of faults (e.g. short-circuits overcurrents) [40]. In addition, the controller provides information related to restoration operations and functions able to locate the faulted phases, determine the status of a device, check tripping counters, etc.

The recloser controller firmware is modified to implement a Man-in-the-Middle (MitM) attack that sniffs Ethernet packets [41]. Specifically, the attack targets the Ethernet packet receiving subroutine, named *tfEtherRecv*, in order to capture the packets of data flowing across the Ethernet network. The modification intercepts the control-flow in the subroutine and copies the received packets to a specific memory location. As a result, an attacker can retrieve the critical information in the received Ethernet packets.

The detection capability of ConFirm on this PowerPC-based firmware is evaluated on the Freescale MPC8308RDB platform with U-Boot as the bootloader. In similar fashion as in case study 1, we select the events with the smallest C.Vs according to Table 1. Because the MPC8308RDB platform has only 4 counters, we select 4 events similar to the ARM-based platform: completed instruction (I), branch (B) instruction completed, load micro-ops completed (L) and store micro-ops completed (S).

The HPC-based signature of the malicious path exhibits large deviations when compared with the signatures of the five valid paths in *tfEtherRecv*. The generated signature is compared with the valid signatures included in the checking window to validate any match. The results are presented in Table 2 (d) for 3 randomly chosen check windows.

The smallest deviation $D(E_x)$ in this scenario is equal to 4.2% due to the LOAD event counts of check window 1. Selecting the appropriate threshold N, for instance 4%, ConFirm differentiates between valid and malicious paths in order to detect the packet sniffing.

Table 2 (e) and (f) show the results of another two firmwares evaluated on the PowerPC-based platform.

4.2 Performance and Storage Overhead

Here we evaluate the performance overhead on the monitored firmware when ConFirm is enabled. The experiment is performed on the presented firmwares and evaluated on the ARM- and PowerPC-based platforms. The runtime performance overhead is mainly due to the execution of extra instructions at each checkpoint: transfer control instructions in the trampoline, HPC handler instructions that read the counters, and ConFirm core instructions that compare the HPC read values with known references. Therefore, the performance overhead tightly depends on the frequency of checks, i.e. the size of a check window.

Figure 6 shows the execution time overhead on the monitored firmwares when different check window sizes are applied. For instance, a check window size of 500 instructions leads to an average execution time overhead of 8.48% on the ARM Cortex-A15 platform and 5.62% on the PowerPC e300c3 platform. For the test cases presented in Section 4.1, the performance overhead for the scenario that includes all the subroutine paths is 14.2% and 7.3% for the ARM and PowerPC case respectively.

The storage overhead of ConFirm is mainly for storing the ConFirm components instructions and trampolines. The requirement for storing the known valid HPC-based signatures can be fine-tuned according to users' requirements on intrusiveness and security benefits.

Assume an HPC vector for a valid path in a check window contains the counts of 5 hardware events. Also suppose that 2 bytes are used to store each counted number (i.e. up to 65536 event counts - large enough for the occurrences of any event within a typical check window). In this case, the storage for the signature of a valid path is 10 bytes. If 10 check windows are applied and there are 10 valid paths in each window, then the required storage is 1 KB. The storage size requirement for ConFirm instructions and the trampolines is less than 10 KB. Consequently, the total storage size requirement in this example is around 10 KB. In the scenario where the firmware image size is 1MB then the storage overhead is translated to 1% of the firmware code size.

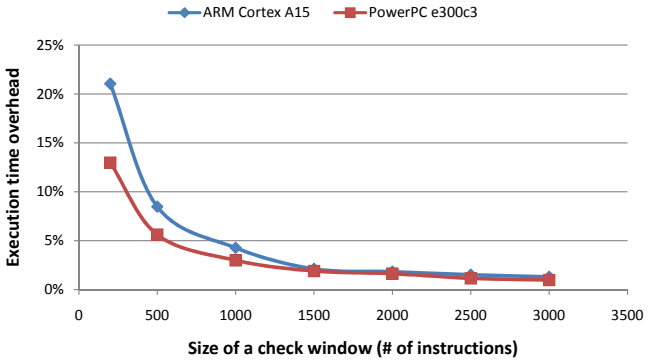


Fig. 6: The execution time overhead when ConFirm is enabled with different sizes of check windows in terms of number of total instructions.

5 Detection with Support Vector Machine-based Machine Learning

5.1 Limitation of Comparison-based Check

The comparison-based HPC signature matching exhibits better performance and accuracy for firmware subroutines that have simple control-flows. However, some firmware may have more complex control-flow and the monitoring subroutines could have many different computational paths, depending on the given inputs and current system state. Consequently, there is a large number of possible valid HPC signatures associated with such monitored subroutines. With the comparison-based matching, all the valid signatures need to be pre-stored in the ROM which significantly increases the storage usage. Moreover, comparing a runtime HPC value with all the stored signatures leads to high performance overhead which may not be acceptable in embedded systems. For example, some subroutines of the two discussed firmwares have more than 50 paths, which means there are more than 50 valid signatures. Therefore, each subroutine requires more than 500 bytes storage and 50 comparisons (worst case scenario) for each check. Storing and matching a subset of the complete set of signatures can reduce the storage and performance overhead but may result in false positives (a valid runtime signature may not be matched).

The problem becomes more complicated when the computational paths include loops with a dynamic number of iterations; the number of valid HPC signatures is enormous. An example is shown in Figure 7. The monitored subroutine is the *Checksum routine for Internet Protocol* from the same ARM-based firmware mentioned in Section 4.1.1. There is a loop in the subroutine calculating the checksum of IP packets, and the number of iterations is determined by the length of the input message. Figure 7 presents the occurrences of the four monitored hardware events INSTRUCTION, BRANCH, LOAD and STORE in 100 executions of the subroutine when different input messages are applied. From the result, we observe that the occurrences of the monitored events are very dissimilar among executions with different inputs. Storing all the possible HPC signatures in the ROM and performing one-to-one comparison with a runtime signature would be infeasible.

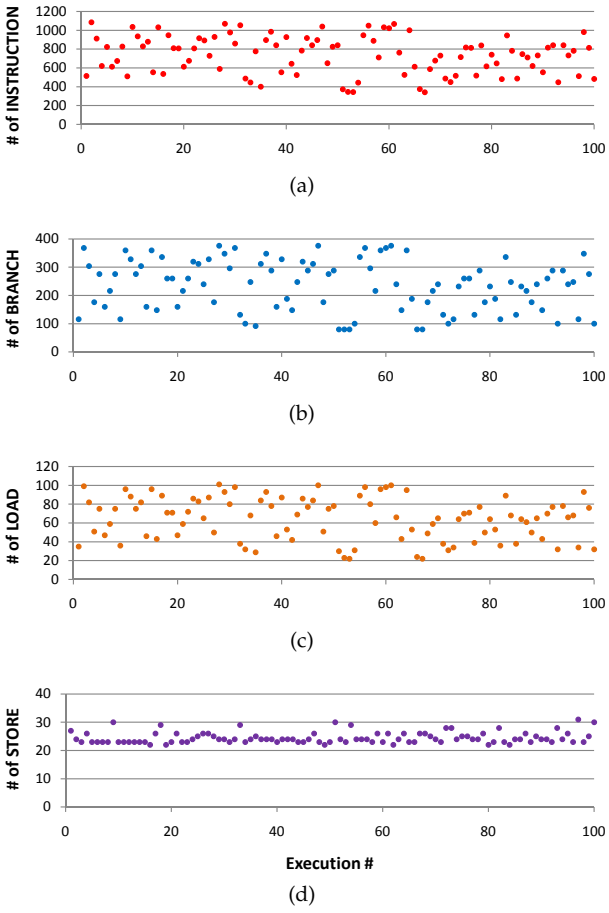


Fig. 7: The occurrences of four monitored hardware events INSTRUCTION (a), BRANCH (b), LOAD (c) and STORE (d) in 100 executions of the same *Checksum routine for Internet Protocol* of the ARM-based firmware which includes a loop in its computational path. A large variation can be observed among different executions when different input arguments are applied.

5.2 Modeling and Classification using One-Class Support Vector Machine

Although the HPC-based signature has large variation for some subroutines, it can be observed that there still exists relatively strong relation among the occurrences of different hardware events. One example is shown in Figure 8. A 3-D dot plot is created with the same HPC data as shown in Figure 7. Each blue dot in Figure 8 denotes an execution of the *Checksum routine for Internet Protocol* subroutine, and the X, Y and Z axes represent the occurrences of events BRANCH, LOAD and INSTRUCTION, respectively. We can observe that the dots of the monitored subroutine are not distributed randomly, instead they gather as a cluster in a certain manner. On the other hand, we modify the original function of this subroutine to simulate a malicious attack that can bypass the checksum. We then execute the malicious subroutine for 100 times and collect the same set of HPC values. The data is plot as red dots in Figure 8 in the same way as mentioned above. Similarly, the dots of the malicious subroutine executions also gather as a cluster. The two clusters are distinctly separated as shown in the 3-

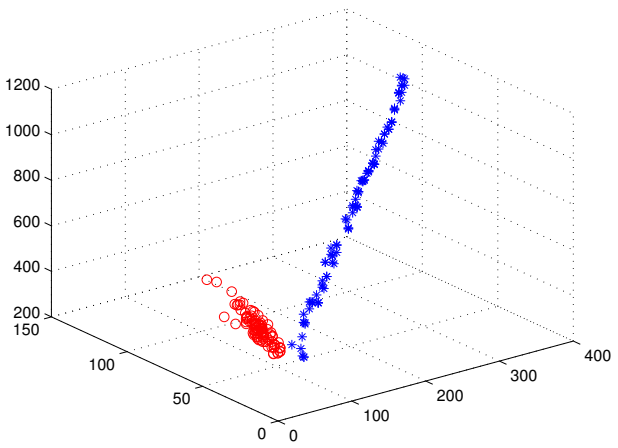


Fig. 8: The dots of normal execution (blue) and malicious executions (red) form two separate clusters in a 3-D plot, giving the possibility to build a model of the normal cluster and rule out all the abnormal data. The X, Y and Z axes in the plot represent the occurrences of events BRANCH, LOAD and INSTRUCTION, respectively.

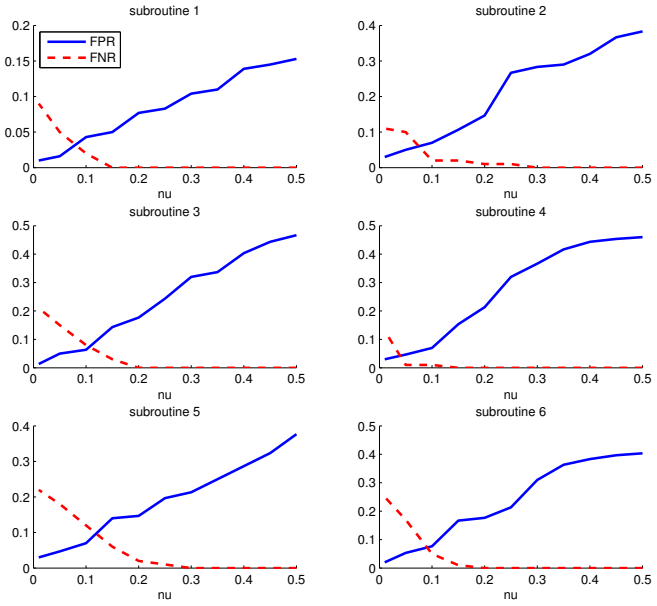


Fig. 9: The FPR and FNR of the OC-SVM classification when different ν values are applied. Subroutine 1-3 are from the ARM-based firmwares and subroutine 4-6 are from the PowePC-based firmwares. The OC-SVM of Libsvm can perform an accurate classification on each subroutine when ν is set properly.

D plot. Therefore, the relations among different monitored HPCs can be used to differentiate normal and abnormal control-flows.

In this work, we use machine learning techniques to automatically extract the relations among different HPCs and build the HPC-based model of the firmware monitored subroutines. This can reduce the number of HPC signatures generated and stored at each platform for the runtime comparison without compromising the detection accuracy. Because it is infeasible to build modules based on the HPC measurements of each potential malicious modification, we use unsupervised one-class machine learning techniques for model building. A one-class machine learning technique is useful in anomaly-based detection because the classifier is trained solely with HPC measurements taken from clean executions. It then classifies test data as similar to or different from the training set. Specifically, we measure the monitored HPCs during the execution of a subroutine and model the characteristics with the One-Class Support Vector Machine (OC-SVM) classifier. SVM can create a non-linear decision boundary by projecting the data through a non-linear function to a space with a higher dimension [42]. More specifically, the data points that cannot be linearly separated in the original space are "mapped" to another space, so called "feature space". In that space, the data points can be separated between classes by a straight hyperplane. When mapped back to the original space, that hyperplane would have the form of a non-linear curve. The main component of the SVM classifier is the kernel function [43]. A kernel function is an algorithm for pattern analysis which can find and study the relations in the given dataset and compute similarity of input data points. In our experiment, we use the non-linear Radial Basis Function (RBF) kernel [44].

The OC-SVM based detection consists of three steps: feature selection, offline training and online classification. During the feature selection, a subset of variables is chosen for model construction. In our technique, the features are the types of hardware events that are monitored during the execution of the firmware. The selection of monitored hardware events are similar to what described in Section 3.4. We choose the events whose occurrences are more repeatable from one execution to another, and more robust against system disturbances. In the offline training phase, the OC-SVM is trained with the data points collected from the executions of the original monitored firmware subroutines to automatically build a "golden" model with the selected features. In the online classification phase, the data points collected at runtime are examined by the trained OC-SVM to determine if they are fitted into the built model. Any data point that is not fitted into the golden model is consider as an anomaly, indicating a malicious execution of the monitored subroutine.

5.3 Detection Capability with OC-SVM

To evaluate the detection capability of the proposed technique, we run the experiments with the same firmwares mentioned in Section 4.1. We use Libsvm [45], an integrated software for support vector classification, regression and distribution estimation, in order to automatically build the model and perform the one-class classification.

For each monitored subroutine, the OC-SVM is first trained with the data collected from the HPC measurements of normal executions to build the model. The subroutines are executed multiple times with different inputs to generate a sufficient number of data points for modelling. The features selected for building models are instructions executed, branch instructions executed, load instructions

executed and store instructions executed, for both ARM- and PowerPC-based firmwares. One of the important steps in the training phase is tuning a set of parameters that is used in OC-SVM. Such parameters have significant impacts on the accuracy of the developed models. For example, the γ parameter defines how far the influence of a single training example reaches, and the ν parameter basically sets an upper bound on the fraction of outliers.

Following the training, we generate two sets of test data points to evaluate the accuracy of the classification of the model built by the OC-SVM. The first set of test data points are generated from the executions of the original subroutines. We applied this set of data points to the OC-SVM model to determine the False Positive Rate (FPR) of the classification. The other set of data points is generated from the executions of the modified subroutines. We perform malicious modifications to the monitored subroutines to simulate different types of attacks. The malicious data points are tested against the built model to determine the False Negative Rate (FNR). The experimental results of the monitored subroutines from each of the six ARM- and PowerPC-based firmwares are shown in Figure 9. For each monitored subroutine, the FPR and FNR are measured when different ν values (0.01 - 0.5) are applied. From the results, we can see that when a proper ν value is chosen, the OC-SVM of Libsvm can perform a very accurate classification with very low FPR and FNR on each subroutine. For example, the classification for subroutine 1 (*Checksum routine for Internet Protocol*) from the ARM-based firmware has both FPR and FNR lower than 5% when ν is set to 0.08.

6 Remote-based Detection with Machine Learning

Using OC-SVM significantly reduces the storage overhead by replacing the large amount of HPC signatures with much smaller models. However, the impact on system performance of running the OC-SVM should be further minimized.

The online phase of the HPC-based detection takes two steps: measuring the occurrences of the monitored HPC events, and performing the analysis on the measured data. In the measurement step, detailed information is obtained with minimal impact on system performance since the occurrences of the events are automatically counted. However, analyzing the measured data (the runtime one-class classification on the HPC data for OC-SVM) still consumes a considerable amount of computing resources. This may become an issue for some real-time platforms with very limited computing capability.

Fortunately, for an HPC-based technique, the analysis is not necessary to be run locally in the monitored system. The analysis steps of the HPC-based checking does not depend on the state of the monitored system after HPC data are collected. Therefore, an effective way to reduce the consumption of local resources is to outsource all the complex analysis to a remote machine which has much higher computing capabilities. Figure 10 shows the structure we propose that provides "centralized" analysis. More specifically, the selected HPC values that are collected locally during the executions of the monitored subroutines of the firmware are sent through the network to a remote machine for further analysis. The OC-SVM runs on the remote machine to perform the classification. The OC-SVM models are pre-generated offline and stored in a database connected to the remote machine. If any anomaly is observed after the classification, the remote machine will send an alert back to the local systems through the network.

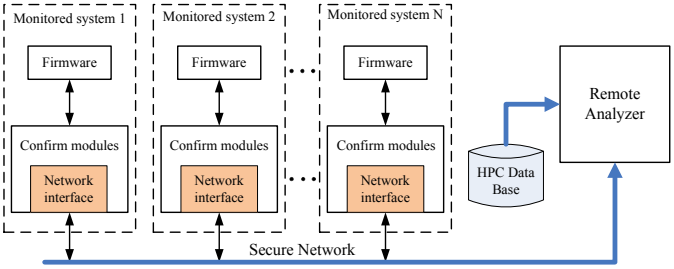


Fig. 10: The structure of remote-based detection. The HPC values measured on a local monitored platform are sent through secure network to a remote machine which has much more computing resources. Both the performance and storage overhead can be significantly reduced on the local monitored platform.

Besides reducing performance overhead, the proposed remote-based detection can further minimize the storage overhead on local platforms. Assume in a smart grid there are N nodes running critical software that needs to be monitored. Each node runs the same software modules with the same hardware configuration (e.g., power meters in the customer premise). With the HPC-based detection enabled, N local databases are required for the whole smart grid while each of them has to contain the complete set of either HPC signatures or SVM models. Each database is actually a duplicate of another. With remote-based technique, only one database is required for the N monitored systems that run the same software modules. There is no extra storage needed at each monitored platform and the storage overhead of the whole system is reduced by N times.

6.1 Performance and Storage Overhead with Remote-based Detection

We evaluate the performance and storage overhead with remote-based detection enabled on both the ARM-based Samsung Exynos Arndale and the PowePC-based MPC8308RDB platforms with the same configurations as described in Section 4. In our implementation, the HPC data are sent from the embedded platforms to the remote machine through a network cable with the Trivial File Transfer Protocol (TFTP) by leveraging the TFTP modules already implemented in U-Boot. The remote machine has the 2.53GHz Intel Core2 Duo CPU running 32-bit Ubuntu 12.04 with 4GB RAM. The performance and storage overhead on the local and remote platforms are summarized in Table 3 (a) and (b), respectively.

On the local platform side, with the OC-SVM and remote-based detection enabled, the perform overheads on the Samsung Exynos Arndale and the MPC8308RDB platforms are 5.1% and 4.7%, respectively. As mentioned before, there is no extra storage needed for signatures or models

TABLE 3: The performance and storage overhead on the local (a) and remote (b) platforms

(a)

	Arndale	MPC8308RDB
Performance overhead	5.1%	4.7%
Storage for signatures	0	0
Network bandwidth	1 KB/sec	1 KB/sec

(b)

Execution time for training	∼2 ms per subroutine
Execution time for classification	∼5 ms per subroutine
Storage for models	∼10 KB per model
Network bandwidth required	1 KB/sec per connection

on local platforms. Assume 100 subroutines are checked every 1 second. For each check, 5 HPC values are measured and sent. Because each HPC value needs 2 bytes, there is a network bandwidth requirement of 1 KB/sec for each monitored platform which is acceptable for most embedded systems nowadays.

On the remote platform side, the training phase takes about 2 ms for building a model of a subroutine while the online classification takes about 5 ms with platform configuration mentioned above. Regarding the storage, a model of a subroutine occupies about 10 KB. The network bandwidth requirement on the remote side depends on the number of local platforms that send the data at the same time. Each connection needs a bandwidth of 1 KB/sec.

7 Related Work

Zimmer *et al.* present three software-based mechanisms for time-based intrusion detection [46]. The applications running on an embedded system are validated by the bounds of execution time of selected code sections. VIPER is a software-only attestation method developed to verify the integrity of peripherals firmware [47]. It is executed on the host machine assuming that the OS on the host CPU is secure and trustworthy during verification. [48] introduces a mechanism to identify abnormal system-wide behaviors by measuring the number of memory accesses to a particular memory region during a time interval. This technique requires modifications to hardware, and has significant noise when the memory usage is caused by network activities or user interactions. OCFMM [49] detects malicious modifications by enforcing control flow integrity in embedded real-time systems. The control flow graph (CFG) of the monitored firmware is loaded into the isolated memory and the check is performed with the granularity of each basic block. This technique also requires hardware modifications and has high performance and storage overheads. NAVIS is an anomaly-detection system that checks memory accesses performed by the Network Interface Card (NIC) on-chip processor [50]. Besides the memory-only profiling of NIC, NAVIS runs inside of the OS and therefore assumes that the OS is trusted. Hu *et al.* have proposed a hardware monitor based attack detection framework in Network Processors (NP) to protect the system from data plane attack by checking the instruction flow of the firmware [51], [52]. Other detection frameworks often require modules that are not present in many embedded designs such as System Management Mode (SMM) [53].

Schellekens *et al.* have studied the integration of a trusted module into a system-on-chip design that lacks embedded reprogrammable non-volatile memory [54]. Doing that, they introduce a cryptographic protocol to achieve an authenticated channel between the trusted module and the external non-volatile memory. It has also been demonstrated how to leverage flash microcontroller units for remote kernel attestation in order to audit application firmware integrity [55]. Furthermore, a study presented a signature verification method in order prevent malicious firmware from being installed on a mouse [13]. The signature-based verification code requires to be inserted into the bootloader. However, the code requires more space than is currently available in the mouse bootloader, highlighting the nature of embedded systems in terms of power, memory, area, timing and other resources constrains.

A static analysis technique compares a suspected-altered PLC firmware to a known good firmware [56]. Static analysis, however, can only detect modifications based on the firmware size and code modules differences. Moreover, Morais *et al.* have developed a verification method which hashes the residing ROM code and compares it with the expected hash value. This technique, however, presents overhead, especially if the main system microcontroller unit does not have a multiplier [57]. Similar to the hash-based mechanism, guards (e.g. checksum functions, obfuscation functions etc.) embedded into the binary program introduce significant resource cost to the embedded system [58].

Epitomizing the techniques on firmware verification, mostly rely on some form of checksum algorithms, such as Cyclic Redundancy Checks (CRC) and cryptographic hash functions. However, Checksum algorithms can be circumvented by reverse engineering their calculation modules [5], [10], [59]. Other verification mechanisms often rely on network connectivity (digital signatures) [60], or even require extra hardware [61]. In comparison with the currently developed detection methods, ConFirm, does not require extra hardware neither has any real-time requirements.

HPCs have been previously studied for security purposes and especially for malware detection and identification. Demme *et al.* have demonstrated how to detect Android malware and Linux rootkits by collecting data of microarchitectural events [62]. NumChecker is a Virtual Machine Monitor (VMM) based framework that can detect rootkits which subvert the control-flow of OS kernels [63]. Using the branch trace store mechanism in HPCs, CFIMon presents how to verify control-flow integrity by detecting attacks such code-reuse attacks [64]. In addition, Ozsoy *et al.* have developed an always-on hardware malware detection engine by measuring low-level hardware events occurrences [65]. BRAIN combines the occurrences of low-level hardware events and network statistics to detect Distributed Denial of Servic (DDoS) attacks [66]. Moreover, a preliminary work on HPC-based malicious firmware detection is presented in [67].

8 Conclusions and Future Work

In this work, we propose ConFirm, a low-cost technique to detect malicious modifications in the firmware of embedded systems by measuring the number of low-level hardware events that occur during execution. In order to count these events, ConFirm leverages the HPCs which readily exist in many embedded processors. We propose a comparison-based technique to detect malicious modifications in firmwares with simple control-flows. For firmwares with more complex control-flows, we use machine learning techniques to automatically extract the relations among different hardware events. This method significantly reduces the number of pre-stored valid HPC signatures without compromising the detection accuracy. Finally, we reduce the consumption of local resources by implementing a remote-based detection mechanism. We evaluate the detection capability and performance overhead of the proposed technique on various types of firmware running on ARM- and PowerPC-based embedded processors. Experimental results demonstrate its practicality and effectiveness.

References

[1] J. Holler, V. Tsiatsis *et al.*, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. Academic Press, 2014.

[2] IDC Research Company, "Intelligent Systems to Exceed $1 Trillion in 2019 as the Market Continues to Disrupt Traditional Industries Including Manufacturing, Energy, and Transportation," [Online]: http://www.idc.com.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[4] C. Miller, "Battery firmware hacking," *Black Hat USA*, pp. 3–4, 2011.

[5] A. Cui, M. Costello, and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, Feb. 2013, pp. 1–13.

[6] A. Costin, J. Zaddach *et al.*, "A Large-Scale Analysis of the Security of Embedded Firmwares," in *Proceedings of the 23rd USENIX Security Symposium*, Aug. 2014, pp. 95–110.

[7] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltsidas, "Implementation and implications of a stealth hard-drive backdoor," in *Proceedings of the 29th Annual Computer Security Applications Conference*, Dec. 2013, pp. 279–288.

[8] C. Heffner, "Reverse Engineering a D-Link Back-door," [Online]: http://www.devttys0.com/, Oct. 2013.

[9] B. Bencsath, L. Buttyan, and T. Paulik, "XCS based hidden firmware modification on embedded devices," in *Proceedings of the 19th International Conference on Software, Telecommunications and Computer Networks*, Sept. 2011, pp. 1–5.

[10] graeme@lolux.net, "Developing a Trojaned Firmware for Juniper ScreenOS Platforms," *The Circle of Lost Hackers*, vol. 0x0d, no. 0x42, pp. 223–246, 2009.

[11] Slashdot, "Backdoor found in TP-Link routers," [Online]: http://tech.slashdot.org/, Mar. 2013.

[12] K. Chen, "Reversing and exploiting an Apple firmware update," *Black Hat, USA*, 2009.

[13] J. Maskiewicz, B. Ellis *et al.*, "Mouse trap: Exploiting firmware updates in usb peripherals," in *Proceedings of the 8th USENIX Workshop on Offensive Technologies*, Aug. 2014, pp. 1–10.

[14] A. Costin, "Hacking MFPs," *The 28th Chaos Communication Congress*, 2011.

[15] pt, "Ooops I hacked My PBX," *The 28th Chaos Communication Congress*, 2011.

[16] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Security Symposium*, Aug. 2011, pp. 1–16.

[17] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, "Take two software updates and see me in the morning: The case for software security evaluations of medical devices," in *Proceedings of the 2nd USENIX Conference on Health Security and Privacy*, Aug. 2011, pp. 6–6.

[18] Z. Basnight, J. Butts, J. Lopez, and T. Dube, "Firmware modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 76–84, 2013.

[19] "OProfile, statistical profiler for Linux systems," [Online]: http://oprofile.sourceforge.net/.

[20] "Hardware performance counters," [Online]: http://en.wikipedia.org/wiki/Hardware_performance_counter.

[21] "Intel 64 and IA-32 Architectures Developer's Manual," [Online]: http://www.intel.com.

[22] X. Wang and R. Karri, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *Proceedings of the 50th Annual Design Automation Conference*, Jun 2013, pp. 79:1–79:7.

[23] "ARM Cortex-A53 Processor Technical Reference Manual," [Online]: http://infocenter.arm.com/.

[24] "4th generation of 32-bit PowerPC microprocessors," [Online]: https://en.wikipedia.org/wiki/PowerPC_G4.

[25] J. Petroni, L. Nick, and M. Hicks, "Automated detection of persistent kernel control-flow attacks," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Oct. 2007, pp. 103–115.

[26] J. Pincus and B. Baker, "Beyond stack smashing: recent advances in exploiting buffer overruns," *IEEE Security & Privacy*, vol. 2, no. 4, pp. 20–27, July 2004.

[27] U. Erlingsson, "Low-Level Software Security, Attacks and Defenses," in *Foundations of Security Analysis and Design IV*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4677, pp. 92–134.

[28] B. Zeng, G. Tan, and G. Morrisett, "Combining control-flow integrity and static analysis for efficient and validated data sandboxing," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011, pp. 29–40.

[29] M. Abadi, M. Budiu *et al.*, "Control-flow integrity," in *Proceedings of the 12th ACM conference on Computer and Communications Security*, Oct. 2005, pp. 340–353.

[30] B. Niu and G. Tan, "Modular control-flow integrity," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun. 2014, pp. 577–587.

[31] Wikipedia the free encyclopedia, "Booting," [Online]: http://en.wikipedia.org/wiki/Booting.

[32] G. Hunt and D. Brubacher, "Detours: binary interception of win32 functions," in *Proceedings of the 3rd conference on USENIX Windows NT Symposium*, Jul. 1999.

[33] "Arndale Board," [Online]: http://www.arndaleboard.org/wiki/index.php/WiKi.

[34] "MPC8308RDB Reference Platform," [Online]: http://www.freescale.com/.

[35] "Cortex-A15 Technical Reference Manual," [Online]: http://infocenter.arm.com/.

[36] "e300 Power Architecture Core Family Reference Manual," [Online]: http://cache.freescale.com/.

[37] "Vxworks: The real-time operating system for the internet of things."

[38] V. Durcekova, L. Schwartz, and N. Shahmehri, "Sophisticated denial of service attacks aimed at application layer," in *Proceedings of the 7th ELEKTRO*, May 2012, pp. 55–60.

[39] "U-Boot," [Online]: http://www.stlinux.com/u-boot.

[40] E. Godoy, A. Celaya, H. J. Altuve, N. Fischer, and A. Guzmán, "Tutorial on single-pole tripping and reclosing," in *Proceedings of Western Protective Relay Conference*, Oct. 2012, pp. 1–21.

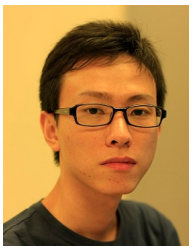
[41] A. Ornaghi and M. Valleri, "Man in the middle attacks," [Online]: https://www.blackhat.com/, 2003.

[42] "Introduction to one-class support vector machines," [Online]: http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/.

[43] J. Shawe-Taylor and S. Sun, "Kernel methods and support vector machines," *lecture notes, June*, 2009.

[44] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers," *Neural networks*, vol. 5, no. 4, pp. 595–603, 1992.

[45] C.-C. Chang and C.-J. Lin, "Libsvm – a library for support vector machines," [Online]: https://www.csie.ntu.edu.tw/~cjlin/libsvm/.
[46] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan, "Time-based intrusion detection in cyber-physical systems," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, Apr. 2010, pp. 109–118.
[47] Y. Li, J. M. McCune, and A. Perrig, "VIPER: Verifying the Integrity of PERipherals' Firmware," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Oct. 2011, pp. 3–16.
[48] M.-K. Yoon, S. Mohan, J. Choi, and L. Sha, "Memory heat map: anomaly detection in real-time embedded systems using memory behavior," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference*, Jun. 2015, pp. 1–6.
[49] F. A. T. Abad, J. van der Woude, Y. Lu, S. Bak, M. Caccamo, L. Sha, R. Mancuso, and S. Mohan, "On-chip control flow integrity check for real time embedded systems," in *Proceedings of the 1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, Aug. 2013, pp. 26–31.
[50] L. Duflot, Y.-A. Perez, and B. Morin, "What if you cant trust your network card?" in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6961, pp. 378–397.
[51] K. Hu, H. Chandrikakutty, R. Tessier, and T. Wolf, "Scalable hardware monitors to protect network processors from data plane attacks," in *Proceedings of the 1st IEEE Conference on Communications and Network Security*, Oct. 2013, pp. 314–322.
[52] K. Hu, T. Wolf, T. Teixeira, and R. Tessier, "System-level security for network processors with hardware monitors," in *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference*, Jun. 2014, pp. 1–6.
[53] F. Zhang, H. Wang *et al.*, "A framework to secure peripherals at runtime," in *Computer Security - ESORICS 2014*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8712, pp. 219–238.
[54] D. Schellekens, P. Tuyls, and B. Preneel, "Embedded Trusted Computing with Authenticated Non-volatile Memory," in *TRUST*, ser. Lecture Notes in Computer Science, vol. 4968. Springer, 2008, pp. 60–74.
[55] M. LeMay and C. A. Gunter, "Cumulative attestation kernels for embedded systems," *IEEE Transactions on Smart Grid*, vol. 3, no. 2, pp. 744–760, 2012.
[56] A. M. J. Garcia, "Firmware Modification Analysis in Programmable Logic Controllers," Ph.D. dissertation, Air Force Institute of Technology, 2014.
[57] D. Morais, J. Lange *et al.*, "Use of hashing in a secure boot loader," 2010, uS Patent 7,676,840.
[58] ARXAN Technologies, "Guarding Technology," [Online]: https://www.arxan.com/why/guarding-technology/.
[59] C. Brunschwiler, "Energy Fraud and Orchestrated Blackouts - Issues with Wireless Metering Protocols (wM-Bus)," 2013.
[60] L. K. Shade, "Implementing Secure Remote Firmware Updates," *Embedded Systems Conference*, 2011.
[61] V. Zimmer and M. Rothman, "Method for performing a trusted firmware/bios update," 2005, uS Patent App. 10/607,367.
[62] 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*, Jun. 2013, pp. 559–570.
[63] 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.
[64] 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*, Jun. 2012, pp. 1–12.
[65] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture*, Feb. 2015, pp. 651–661.
[66] 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 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems*, Jan 2016, pp. 587–588.
[67] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *Proceedings of the 34th IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2015, pp. 544–551.



Xueyang Wang (S'14) received the B.S. degree in Automation from Zhejiang University, Zhejiang, China, in 2008 and the M.S. and Ph.D. degrees in Computer Engineering and Electrical Engineering from Tandon School of Engineering, New York University, Brooklyn, NY, USA, in 2010 and 2015, respectively.

His research interests include secure computing architectures, virtualization and its application to cyber security, hardware support for software security and hardware security.

Charalambos Konstantinou (S'11) received the five-year diploma degree in Electrical and Computer Engineering from the National Technical University of Athens, Athens, Greece. He is currently pursuing the Ph.D. degree in Electrical Engineering with the New York University Tandon School of Engineering, New York. His interests include hardware security with particular focus on embedded systems and smart grid technologies.

Michail Maniatakos received the B.Sc. degree in Computer Science and the M.Sc. degree in Embedded Systems from the University of Piraeus, Piraeus, Greece, in 2006 and 2007, and the M.Sc. and M.Phil. degrees and the Ph.D. degree in Electrical Engineering from Yale University, New Haven, CT, USA in 2009, 2010, and 2012.

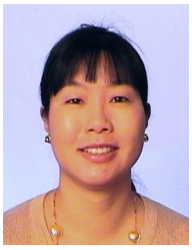
He is an Assistant Professor of Electrical and Computer Engineering at New York University (NYU) Abu Dhabi, Abu Dhabi, UAE, and a Research Assistant Professor at the NYU Tandon School of Engineering, Brooklyn, NY, USA. He is the Director of the MoMA Laboratory (nyuad.nyu.edu/momalab), NYU Abu Dhabi. His research interests, funded by industrial partners and the U.S. Government, include robust microprocessor architectures, privacy-preserving computation, as well as industrial control systems security. He has authored several publications in IEEE transactions and conferences, and holds patents on privacy-preserving data processing.

Dr. Maniatakos is currently the Co-Chair of the Security track at IEEE International Conference on Computer Design (ICCD) and IEEE International Conference on Very Large Scale Integration (VLSI-SoC). He also serves in the technical program committee for various conferences, including IEEE/ACM Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD), International Test Conference (ITC), and International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES). He has organized several workshops on security, and he currently is the faculty lead for the Embedded Security challenge held annually at Cyber Security Awareness Week (CSAW), Brooklyn, NY, USA.

Ramesh Karri received the Ph.D. degree in Computer Science and Engineering from the University of California at San Diego, La Jolla, CA, USA in 1993.

He is a Professor of Electrical and Computer Engineering at Tandon School of Engineering, New York University, Brooklyn, NY, USA. His research and education activities span hardware cybersecurity: trustworthy ICs, processors, and cyberphysical systems; security-aware computer-aided design, test, verification, validation, and reliability; nano meets security; metrics; benchmarks; and hardware cybersecurity competitions. He has over 200 journal and conference publications including tutorials on trustworthy hardware in IEEE COMPUTER (two) and PROCEEDINGS OF THE IEEE (five).

Dr. Karri was the recipient of the Humboldt Fellowship and the National Science Foundation CAREER Award. He is the area director for cyber security of the NY State Center for Advanced Telecommunications Technologies at Tandon School of Engineering, New York University; Cofounded (2015-present) the Center for Cyber Security (CCS) (http://crissp.poly.edu/), co-founded the TrustHub (http://trust-hub.org/) and founded and organizes the Embedded Security Challenge, the annual red team blue team event at NYU, (http://www.poly.edu/csaw2014/csaw-embedded). His group's work on hardware cybersecurity was nominated for best paper awards (ICCD 2015 and DFTS 2015) and received awards at conferences (ITC 2014, CCS 2013, DFTS 2013 and VLSI Design 2012) and at competitions (ACM Student Research Competition at DAC 2012, ICCAD 2013, DAC 2014, ACM Grand Finals 2013, Kaspersky Challenge and Embedded Security Challenge). He co-founded the IEEE/ACM Symposium on Nanoscale Architectures (NANOARCH). He served as program/general chair of conferences including IEEE International Conference on Computer Design (ICCD), IEEE Symposium on Hardware Oriented Security and Trust (HOST), IEEE Symposium on Defect and Fault Tolerant Nano VLSI Systems (DFTS) NANOARCH, RFIDSEC 2015, and WISEC 2015. He serves on several program committees (DAC, ICCAD, HOST, ITC, VTS, ETS, ICCD, DTIS, WIFS). He is the Associate Editor of the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY (2010-2014), IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (2014-present), ACM Journal of Emerging Computing Technologies (2007present), ACM Transactions on Design Automation of Electronic Systems (2014-present), IEEE ACCESS (2015-present), IEEE TRANSACTIONS ON EMERGING TECHNOLOGIES IN COMPUTING (2015-present), IEEE DESIGN & TEST (2015-present), and IEEE EMBEDDED SYSTEMS LETTERS (2016-present). He is an IEEE Computer Society Distinguished Visitor (2013-2015). He is on the Executive Committee of IEEE/ACM Design Automation Conference leading the cybersecurity initiative (2014-present). He has delivered invited keynotes and tutorials on hardware security and trust (ESRF, DAC, DATE, VTS, ITC, ICCD, NATW, LATW).



Serena Lee has been with Con Edison for 25 years and is currently Project Manager for the R&D Department. Her overall responsibilities include promoting new innovations and managing research, development and demonstration projects for Con Edison electric system. Serena started in Con Edison as a management intern and spent the majority of her career in electric distribution. Serena has spent more than a decade in various roles within the company managing maintenance and engineering organizations overseeing system reinforcement, reliability improvements and customer projects. Prior to her role in Research and Development, she held the position of senior engineer in distribution engineering where she was responsible for launching the development of Con Edison's network distribution systems reliability prediction and optimization programs. She has also worked extensively on Con Edison Third Generation Planning studies to formulate new design and operation strategies to meet Con Edison's future loads.

Serena Lee graduated from State University of New York at Stony Brook with a Bachelor of Engineering degree in Electrical Engineering and from NYU Tandon School of Engineering with a Master of Science in Electrical Engineering. She is also a certified PMP.

Paul V. Stergiou holds a B.E. in Electrical Engineering from City College of New York and a Distribution System Engineering Certificate from Power Technologies.

He is currently the Department Manager of Distribution Engineering's SCADA and Unit Substation sections at Con Edison of New York. The department is responsible for the design and implementation of electric distribution SCADA (Supervisory Control and Data Acquisition) systems and 4kV Unit and Multibank Substations. In his 28 years of service with Con Edison, Mr. Stergiou has worked on a number of projects that enhanced the electric distribution system, including SCADA, Unit Substations, Distribution Automation System, GPS Phase Matching System, Reactance-To-Fault, Intelligent Autoloops, Power Quality, interconnection of PVs and DGs, Power Line Carrier, and Voltage-Var-Control (VVC). He has designed numerous overhead and underground distribution systems, including underground networks, overhead autoloops, and URD. His recent work is focused on Distribution Automation, enhanced SCADA systems, Storm Hardening Projects, 4kV Unit and Multibank Substations and the safe and reliable interconnection of PV, DG and micro grids. Mr. Stergiou holds patent number 8180481 "Autoloop system and method of operation." Mr. Stergiou also earned a global honor for a GPS-Based System he co-invented to bring a new substation on line after the 9/11 attack. Mr. Stergiou was named to the "50 Key IT Players in Energy" Annual Honors list in 2003 for this invention.

Steve Kim is the Manager of the Cyber Action Team within Con Edison Corporate Security. The Cyber Action Team was formed as part of Corporate Security in 2013. It was created as a cyber-investigative team with standards, methodologies, and processes designed to address cyber risks to Con Edisons critical infrastructure. The Cyber Action Team has the ability to forensically identify, investigate and mitigate cyber threat incidents with a goal of maintaining business continuity. Steve Kim earned a Doctorate degree in Computing from Pace University. He is an Adjunct Professor in Cyber Security and Digital Forensics at various Universities in NYC area.