PHYLAX: Snapshot-based Profiling of Real-Time Embedded Devices via JTAG Interface

Charalambos Konstantinou*, **Eduardo Chielle**†, **Michail Maniatakos**†
*Electrical and Computer Engineering, Tandon School of Engineering, New York University
†Electrical and Computer Engineering, New York University Abu Dhabi
E-mail: {ckonstantinou, eduardo.chielle, michail.maniatakos}@nyu.edu

***Abstract*—Real-time embedded systems play a significant role in the functionality of critical infrastructure. Legacy microprocessor-based embedded systems, however, have not been developed with security in mind. Applying traditional security mechanisms in such systems is challenging due to computing constraints and/or real-time requirements. Their typical 20-30 year lifespan further exacerbates the problem. In this work, we propose PHYLAX, a plug-and-play solution to detect intrusions in already installed embedded devices. PHYLAX is an external monitoring tool which does not require code instrumentation. Also, our tool adapts and prioritizes intrusion detection based on the requirements of the underlying infrastructure (power grid, chemical factory, etc.) as well as the computing capabilities of the target embedded system (CPU model, memory size, etc.). PHYLAX can be employed on any legacy device which incorporates a JTAG interface. As a case study, we present the inclusion of PHYLAX on a power grid recloser controller.**

I. Introduction

Over the past fifty years, Industrial Control Systems (ICS) have been entrusted to ensure safe and reliable operation of critical physical processes. Even though ICS devices and protocols have many inherent vulnerabilities, since security was not a concern during their standardization, the fact that ICS technology was hosted on dedicated infrastructure rendered it immune to cyber threats. This has changed in the last twenty years, as cost pressures are dictating convergence of conventional Information Technologies (IT) with ICS [1].

Installed devices "smart" but not secure: A quick search using SHODAN, browsing through the literature and the vendor brochures reveals that the IT convergence with ICS is funneled through the inclusion of microprocessor-based embedded systems. Most of these devices were developed decades ago without security in mind and also designed with a lifespan of 20-30 years. Thus, insecure embedded devices will remain a major threat for many years.

Replacement impractical: Even if we assume that the newly installed embedded devices are well-secured, it is impractical and resource-intensive to replace those already in place. In addition to the lifetime of these devices, an industry mantra still plagues the mindset of some executives: "Run the equipment until it dies" [2]. Scrapping legacy systems and replacing them with modern ones may involve risks due to customized specifications, unpredictable costs, etc.

Software upgrade impractical: An interim solution could potentially be a software upgrade of existing devices to include security features [3]. This solution may be more challenging than replacing the devices: First, vendor collaboration is required. The vendor may be out of business, the product may be discontinued or the vendor may refuse to collaborate. Second, the device could be incapable of improvement, due to limited computational capabilities, insufficient free space, etc. Third, even if an improved firmware is developed, the utility needs to have proper updating mechanisms. For third parties this is also an extremely challenging task as source code is not typically available. Also, the performance overhead is often too large to be deployed in real-time applications [4].

Hardware-assisted approaches demonstrate significant performance improvement without the requirement of code instrumentation. Nevertheless, such schemes require hardware logic to be integrated in the processor [5]. Various schemes have been proposed utilizing debug interfaces [6]. Such approaches, however, rely on special debug and trace modules provided by chip-specific and modern debug architectures not typically available in legacy devices [7].

Given all the above, an interim solution which addresses the security concerns of critical infrastructure *as they exist today* is needed. Hence, in this work, we propose PHYLAX[1], an external monitoring tool with the following key features:

- can be installed on any JTAG-enabled device,
- prioritizes and adapts to any control process,
- does not require instrumentation or vendor collaboration.

PHYLAX is a ready-to-deploy JTAG-based monitoring and detection module which can be attached to the host processor of the embedded device. The module inspects memory and registers, and inserts breakpoints to detect abnormal behavior (Section II). In order to enable authorized access to JTAG port and protect JTAG data, various schemes have been introduced [8]. The proposed work can be used in conjunction with all these techniques. We demonstrate PHYLAX's capabilities on a power grid case study in Section III. It should be emphasized, and will be further discussed in Section IV, that the massive deployment of PHYLAX to all embedded devices of the infrastructure is equally unrealistic. Instead, potential applications include selective deployment to very critical nodes, and using PHYLAX as a forensics tool for rapid diagnosis and intruder tracing after a cyberattack.

[1]Spawning from ancient Greek, the term phylax (φύλαξ) is commonly translated as "guard" or "watcher".

978-3-9819263-0-9/DATE18/©2018 EDAA

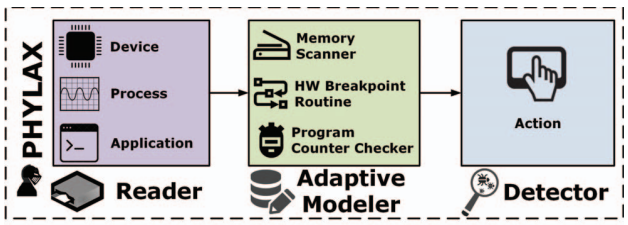

Fig. 1: PHYLAX architecture.

II. PHYLAX Methodology

A. Prerequisites

JTAG-enabled device: IEEE 1149.1, known as JTAG, is the industry standard for the test access port and the boundary scan architecture. Besides testing purposes, JTAG is widely used for in-circuit debugging and firmware programming.

Specific debugging features: Debug features are configurable by chip designers and can be classified into invasive and non-invasive. Invasive features require to halt the processor or change significantly the execution flow. Non-invasive features have none or very little effect on the program flow. Among the available features, PHYLAX relies on those that exist in most debugging designs, which are a) internal register access (invasive), b) memory access (non-invasive), and c) placement of hardware breakpoints (non-invasive).

The selection of JTAG as the extraction medium and the subset of debugging features required were driven by the need to make PHYLAX widely applicable to legacy devices. Finally, we assume that the PHYLAX-hosting platform has typical computational capabilities (e.g., a Raspberry PI).

B. Threat Model

PHYLAX's design focuses on detecting firmware modifications running on bare metal hardware. This is often the case for many real-time embedded devices with time-critical latency requirements such as those required in ICS. Such systems boot directly monolithic, single-purpose software, and the firmware tasks are executed in a single-threaded and infinite loop. Our threat model allows an adversary to inject or/and alter firmware code and data in such devices in a way that introduces execution of malicious code or circumvention of firmware functions in the code flow execution.

C. PHYLAX Architecture

PHYLAX architecture is designed to be adaptive and customized to fit different application scenarios. Our tool is configured based on a layered structure shown in Fig. 1. Each layer receives information from the previous layer and forwards the results to the following layer.

Reader: The reader is in charge of capturing information based on three input categories:

a) Device-specific parameters: The memory size S of the system, the memory and register access rate via the JTAG port (R_m and R_r), the number of available hardware breakpoints B, and the time to add or remove (t_B) a breakpoint.

b) Process-specific parameters: The operation frequency of the monitoring process f is related with the underlying infrastructure in which PHYLAX is deployed. For example, in the case of the power grid, f is the utility frequency (60Hz).

c) Application-specific parameters: First, the check interval of the suspicious mode C_{susp}. *Suspicious mode* refers to a secondary form of operation for PHYLAX, where intrusive debugging is allowed. While PHYLAX's main operating mode is to remain transparent, specific triggers may warrant intrusive inspection. An example is preventing instruction execution from the data segment: a buffer overflow attack would send instructions through data input and would redirect execution to these instructions. A carefully placed hardware breakpoint can detect this and halt the processor while triggering corrective action. Second, given the process real-time requirements, PHYLAX may not be able to scan the whole memory region within f. Hence, regions can be classified into different types τ and with different number of bytes scanned per process cycle b_c (up to R_m/f bytes can be read per cycle).

Adaptive Modeler: The next layer reads the parsed parameters and generates the monitoring routine. The algorithm is based on: *(a)* a memory scanner, which continuously searches for suspicious and potentially illegitimate memory modifications, *(b)* a breakpoint routine, which utilizes hardware breakpoints in suspicious memory contents, and *(c)* a program counter checker which verifies if the memory address of the next instruction is valid.

a) Memory Scanner (MS): The device hosting PHYLAX initiates monitoring after copying the golden firmware which resides in the embedded chip memory, creating a reference database. This assumes that the malware has not altered the firmware stored in the non-volatile memory used to boot the device. In case this assumption is challenged, a golden firmware has to be retrieved from an external source (e.g. vendor website). During monitoring, the MS continuously extracts memory and inspects the data. Depending on the monitoring process features and the memory access rate R_m, prioritizing which type of memory region τ_i should be extracted more frequently is essential; transferring the whole memory at once is typically infeasible given the real-time process requirements.

Due to the application features, we classify the memory regions into *immutable* and *mutable*. The immutable consists of read-only code and data (ROM) such as application-specific content that should not be modified. Mutable content includes modifiable memory such as random access memory (RAM). After every memory scan cycle the acquired memory is examined. PHYLAX reads the data corresponding to the immutable content and refers to the database to retrieve the reference memory acquired during initialization. Regarding the acquired mutable data, PHYLAX checks if instructions (code) reside in areas that are marked as non-executable. In such scenario, PHYLAX sets a hardware breakpoint to that suspicious location. If the execution flow encounters breakpoints in those modules PHYLAX will alert about possible malware content.

b) Hardware Breakpoint Routine (HBR): Breakpoint instructions cause processors to enter a debug state allowing

investigation of the program after an address is reached. Although limited in number, they already exist in hardware and can be utilized in any type of memory. Most importantly, they do not alter the executed code, stack, or any target resource, thus they can be added without instrumenting the firmware.

The HBR is triggered when the MS identifies mutable memory that matches instructions. Then, a hardware breakpoint is set to that location causing detection of unexpected behavior if a malware deviates the execution to that address. Due to limited availability of hardware breakpoints, their management is critical. Thus, in the memory update cycle which mutable content matches instructions, PHYLAX marks that address as suspicious and adds it into a list. If the suspicious content has no breakpoint set and there available ones, a breakpoint is assigned. In the next update cycle, if any of the previous suspicious addresses no longer point to valid instructions then the breakpoint is removed. If all breakpoints are in use, PHYLAX places the suspicious address in a waiting list.

c) Program Counter Checker (PCC): This mechanism examines if the PC is within the code area. This evaluation, although valuable, needs to be performed moderately as the internal register access is intrusive. PHYLAX activates PCC only when entering suspicious mode. The scenario in which the PCC is triggered at deterministic intervals opens up an opportunity for an adversary to bypass the PCC. Thus, PHYLAX implements a PCC subroutine which randomly sets the PCC check interval. Once the PC is extracted, it is checked for validity, and, depending on the result appropriate actions are taken.

Detector: The last layer is the detector which checks if any mechanism detected a potential violation. In this comparison-based phase, the acquired runtime JTAG data are analyzed. In case of intrusion, an alarm will be triggered by PHYLAX and appropriate actions, specified by the integrator, will be initiated. Possible reactions include disabling/rebooting the embedded device or restoring memory contents to expected values. The presented work focuses on the detection part.

III. Case Study: Power grid monitor

A. Power Grid Node

A node of a typical physical grid architecture consists a bus protective element of a relay or recloser controller and a Circuit Breaker (CB). The controller essentially manages the status signal $B_k(t)$ for the CB k. The analyzing step of the controller has a minimum response time of two power cycles, i.e., for a 60Hz system the CB trips if the fault current exceeds the minimum trip value for more than two cycles (33ms) [9].

B. Experimental Setup

1) Hardware-In-The-Loop (HITL) testbed: We implemented a prototype of PHYLAX monitoring a recloser controller in a HITL testbed. The HITL environment is used for the cybersecurity assessment of PHYLAX and to verify the symbiotic relationship of the hardware equipment. PHYLAX attaches to the JTAG of the tested embedded CB controller. The controller features a 32-bit ARM Cortex-M4 168 MHz core. Table I presents information about the tested device.

TABLE I: Monitoring power grid using PHYLAX.

Input Category	Parameter	Value
Device	S	1Mb (Flash), 192Kb (RAM)
	R_m	7.4585 bytes/ms
	R_r	193.4ms
	B	6
	t_B	3.5ms
Process	f	60Hz
Application	C_{susp}	0
	M	(See Table II)

TABLE II: Memory regions classification.

Priority	Type τ	Size	Bytes/ cycle b_c
Critical	Immutable	40 B	40
High	Immutable	6 kB	64
Medium	Mutable	100 kB	64
Low	Immutable & Mutable	1 MB	16

2) Tested malware cases: The implemented modifications follow the firmware Trojan taxonomy specialized for smart grid devices [10]. Since we focus on the operation of recloser controllers, all the designed modifications can be applied to CBs. The insertion phase of the modifications can be either before or after the deployment of the embedded system via JTAG port, chip-off forensics, and communication links.

In **Case 1** a change of a single instruction starts an infinite loop thread – Denial-of-Service (DoS) – after the recloser over-current condition is triggered. The **Case 2** is an always-on, functional modification which alters the recloser minimum trip setting. **Case 3** is a time bomb modification; it injects code to the memory data and once the breaker trips, the execution is deviated to that address. **Case 4** denies memory service by continually accessing memory content, downgrading system performance or causing DoS. **Case 5** is similar to Case 1, however, a greater number of instructions are modified to perform a more meaningful malicious task. Finally, in **Case 6** the scenario is similar to Case 3, however, it is adapted to perform malicious tasks with larger in size injection of code.

C. PHYLAX Configuration

MS: Although memory can be extracted via JTAG at real-time, i.e., when processor is running, the average access rate of $R_m = 7.4585$ bytes/ms is low due the JTAG serial access to memory. Hence, PHYLAX configures MS to collect memory in small chunks per monitoring scan cycle (33.3ms for 60Hz). Since we cannot extract the whole memory in 33.3ms, we categorize memory into priority levels, as shown in Table II.

HBR: HBR utilizes all six breakpoints of Cortex-M4 and implements a stack as a waiting list. When breakpoints are in-use, the last address identified with suspicious data is pushed to the stack. Once a breakpoint is available, the last element added is popped and the breakpoint is set to its address.

PCC: Due to the invasive nature of accessing processor internal registers, PCC requires considerably more time to extract information. The total average time required for halting the processor, extracting the PC, and resuming execution for the tested device is 193.4ms. Thus, based on our case study requirements assuming a process cycle of 33.3ms, PCC can not be used (C_{susp} is set 0 in Table I).

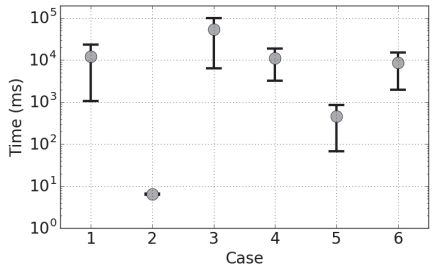

Fig. 2: Time to detect each malware case via MS and HBR.

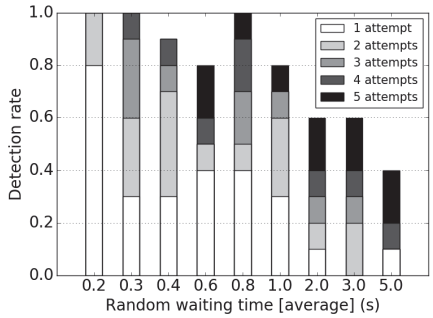

Fig. 3: PCC detection rate per random interval waiting time.

D. Results

In this part, we evaluate the capabilities of PHYLAX. Malware targeting modifications of immutable memory are captured by MS (Cases 1, 2, 4, and 5). HBR detects modifications in the data area of the memory (Cases 3 and 6).

Fig. 2 presents the average value and the standard deviation of the time required by MS and HBR to detect each malware. While PHYLAX detects modifications in critical memory in $<$10ms (Case 2), other cases are identified with greater latency. PHYLAX detects Case 3 with the highest latency due to the large size of the mutable area to be read, and the existence of a single instruction in the data part of the memory. Still, PHYLAX is able to detect all tested cases within the requirements of CBs operation: actual tripping of a CB is an infrequent event, e.g., medium-voltage CBs ($600V$-$15kV$) trip/close around 2000 times annually [11].

In this case study, PCC is disabled since a PC check (193.4ms) cannot be performed in the 33ms cycle. To assess its effectiveness, we remove the 33ms requirement and perform experiments with random check intervals. Since PCC detects illegitimate control flow, it can essentially detect Cases 3 and 6. Fig. 3 summarizes the detection rate of Case 3 with respect to the waiting cycle of PCC. For each scenario, the CB is tripped up to 5 times and the simulation is repeated 10 times for each cycle. Focusing on the 200ms waiting interval, PCC detects the malware in the first attempt for 80% of the simulations, and for the other 20% PCC detects it in the second attempt. As the waiting time interval increases, the detection rate deteriorates, as expected. The results reveal the careful trade-off between the detection rate and the random waiting time, which must be set accordingly for different infrastructure processes.

IV. Limitations and Discussion

Chassis intrusion required: JTAG pins are not exposed to the device external interface. Thus, the integrator needs to physically open the device, identify and tap on the JTAG pins.
JTAG accessibility: Vendors may attempt to destroy or lock JTAG pins in order to prevent unauthorized access. While we have not seen such methods in tested devices, we expect newer devices to include such protection mechanisms.
Data annotation: While PHYLAX is developed to sidestep vendors, prioritization of monitoring relies on data annotation. This can be simple for power grid monitoring (i.e., check the trip value), but more challenging for non-linear processes.
Limited dataset: The malware are the only ones we could port from [10], given the capabilities of the tested device.
Deployment: Deploying PHYLAX across the board would be prohibitive. Instead, PHYLAX needs to be selectively deployed to vulnerable nodes. Also, PHYLAX can act as a post-attack forensics analysis tool. In a cyberattack scenario to the US grid, the organizations that operate the system [1] must prudently allocate their resources towards recovery. The government would have to investigate the attack without relying on immediate support from utilities and vendors. In that scenario, PHYLAX can prove an invaluable tool.

V. Concluding Remarks

Although ICS security is growing and newer devices incorporate security features, proper security may require years to culminate. In this work, we propose PHYLAX, a monitoring tool which can be used *today* on installed legacy devices controlling real-time processes. PHYLAX can adapt and prioritize based on the underlying process and the computing constraints.

Acknowledgment

This work was supported by the NYU Abu Dhabi Global PhD Fellowship program.

References

[1] "Rapid Attack Detection, Isolation and Characterization Systems (RADICS)." [Online]: http://www.darpa.mil/.
[2] C. Eaton, "Hacked: Energy industry's controls provide an alluring target for cyberattacks." [Online]: http://www.houstonchronicle.com/, 2017.
[3] M. Abadi, "Control-flow integrity," in *Proceedings of the Computer and Communications Security conference*, pp. 340–353, ACM, 2005.
[4] J. Newsome and D. Song, *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software*. Internet Society, 2005.
[5] D. Arora *et al.*, "Hardware-assisted run-time monitoring for secure program execution on embedded processors," *IEEE Transactions on VLSI Systems*, vol. 14, no. 12, pp. 1295–1308, 2006.
[6] Z. Guo *et al.*, "Control-flow checking for intrusion detection via a real-time debug interface," in *SMARTCOMP*, pp. 87–92, IEEE, 2014.
[7] X. Zhai, "A method for detecting abnormal program behavior on embedded devices," *IEEE TIFS*, vol. 10, no. 8, pp. 1692–1704, 2015.
[8] S. Kan, "Echeloned ijtag data protection," in *AsianHOST*, IEEE, 2016.
[9] R. Patterson *et al.*, "A microprocessor-based digital feeder monitor with high-impedance fault detection," in *47th Annual Conference for Protective Relay Engineers, Texas, USA*, 1994.
[10] C. Konstantinou *et al.*, "Taxonomy of firmware trojans in smart grid devices," in *PES General Meeting (PESGM)*, pp. 1–5, IEEE, 2016.
[11] Hydroelectric Research and Technical Services Group, *Maintenance of power circuit breakers*, vol. 3-16. USBR, 1999.