Work-in-Progress: Enabling Secure Boot for Real-Time Restart-Based Cyber-Physical Systems

Sena Hounsinou*, Vijay Banerjee*, Chunhao Peng*, Monowar Hasan†, Gedare Bloom*
*University of Colorado Colorado Springs, Colorado Springs, CO, USA
†Wichita State University, Wichita, KS, USA

***Abstract*—Several cyber-physical systems use real-time restart-based embedded systems with the Simplex architecture to provide safety guarantees against system faults. Some approaches have been developed to protect such systems from security violations too, but none of these approaches can prevent an adversary from modifying the operating system or application code to execute an attack that persists even after a reboot. In this work, we present a secure boot mechanism to restore real-time restart-based embedded systems into a secure computing environment after every restart. We analyze the delay introduced by the proposed security feature and present preliminary results to demonstrate the viability of our approach using an open-source bootloader and real-time operating system.**

***Index Terms*—Secure Boot, TEE, U-Boot, RTEMS**

I. Introduction

Real-time embedded systems (RTES) are used in many application domains to control cyber-physical systems (CPS). Many CPS are modeled using the Simplex architecture [1], [2]. The architecture was initially developed to provide a framework for high-reliability and fault-tolerance in rapidly evolving mission-critical systems. In general, the architecture comprises a complex partition and a safety partition (see Figure 1). The complex unit typically integrates system timers and a monitoring unit to detect a fault and trigger a system reset, when necessary. Although this partition is designed to efficiently execute all the functionalities that constitute the mission of the system, it is usually too complex to fully verify.

The safety partition is managed by a *fully verified* controller designed with state-dependent constraints to guarantee that the system will not reach certain predetermined states that are deemed unsafe. The safety unit is located on separate hardware that is not connected to the outside world. The goal of this partition is to allow the system to recover from a software fault originating from the complex unit, using a read-only memory which has an unaltered image of the operating system (OS).

Although the Simplex architecture guards against faults, it does not secure the system against security threats targeting restart-based CPS. To address this problem, a few approaches have been presented recently. They focus on diversifying configuration files, randomizing the location of the executable code, and randomizing hardware to prevent the attacker from using the same method to attack the system after a restart.

This work is supported in part by NSF grants OAC-2001789, CNS-2046705, and Colorado State Bill 18-086.

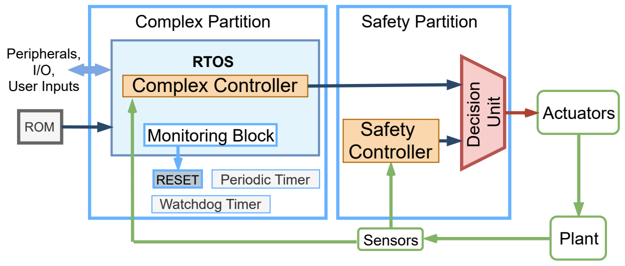


Fig. 1. Architecture of Restart-Based Systems.

These mechanisms however do not prevent a skillful attacker from controlling the system beyond a restart: an adversary can modify the OS or application codes and execute an attack that persists even after a reboot.

In this work, we present an approach that restores the system into a secure computing environment after every restart, by *integrating the secure boot and chain of trust features in the booting sequence*. Secure boot prevents booting a corrupted OS or application during the boot process, by building a chain of trust located in the safety unit. The "root of trust" module verifies a trusted boot code that initializes the hardware. Next, the bootloader proceeds to verify the OS which in turn authenticates the application(s). At each step, if the component is successfully verified, it is added to the chain of trust. Otherwise, the system locks itself and become unusable. In that way, every OS and application image is authenticated prior to being loaded and available for use.

Our contributions are summarized as follows:

- We propose a secure boot mechanism for restart-based real-time systems.
- We provide a detailed analysis of the delay introduced by the proposed security feature and determine its bounds.

II. System and Adversary Models

System Model. We consider a real-time CPS with strict safety and temporal requirements. Examples of such systems include industrial process control systems or avionic systems. The RTES invokes a system-wide restart from a hardware root of trust [3] to restore the plant into a safe operational window if one of the following occurs: (1) the monitoring unit has detected an attack, (2) the periodic timer indicates that the predefined interval between two restarts has elapsed or (3) the watchdog timer has detected a failure of a critical component.

We consider that the system is composed of n periodic tasks $\tau_1, \ldots, \tau_n$ and a sporadic restart task τ_r. Let h be the hyperperiod of the periodic tasks (i.e., the least common multiple of their periods). Each task produces multiple instances and $\tau_{i,k}$ represents the k-th instance of the task τ_i. All tasks are executed on the same processor under a fixed priority preemptive scheduling policy where i is the priority of τ_i. τ_r is the highest priority task in the system. Each periodic task τ_i is characterized by a tuple (C_i, T_i, ϕ_i). C_i is the worst case execution time of τ_i. T_i represents the period (and relative deadline). ϕ_i is the release time of the first job $\tau_{i,0}$. For the restart task, T_r represents the minimum time between two restarts and C_r is the duration of a single restart operation after which any job that has not been completed prior to the restart is executed in its entirety (i.e., jobs that have been preempted are re-executed).

Adversary Model. We assume that the safety partition of the RTES is isolated and cannot be accessed by any user or remote attacker. Therefore, the safety unit, including the root of trust, is out of reach of the attacker. In contrast, the complex partition is connected to user input interfaces and to the peripherals through the system network. In this work, we specifically focus on an adversary that manipulates the OS image or application in the complex partition, with the intention of taking control of the system. Finally, we assume that the attacker has no physical access to the system. Therefore, any malicious action taken against the system is done remotely using software manipulations.

III. Proposed Design

Our goal is to increase the security of CPS by providing a mechanism that guarantees that the system is tamper-free right after each restart without impacting the proper operation of the system (that is, all tasks can still execute without missing their deadlines, even in the presence of restarts). A tamper-free computing environment is usually achieved by integrating a secure booting sequence at power up. Thus, to achieve our goal, we need to integrate a security module to provide the functionalities needed to implement secure boot in the current architecture. We describe the module in Section III-A.

In addition, we analyze the impact of the proposed mechanism on the system's performance; by implementing the secure boot feature, the restart task will require more time to execute and therefore may affect the schedulability of the system. Thus, in addition to providing the architectural resources, our next challenge is to ensure that the integration of secure boot sequence does not degrade the system's performance. We study these implications in Section III-B.

A. Secure Boot for Simplex-based Embedded System

Generally, secure boot provides four features: 1) protected access-control capabilities, 2) integrity measurement, 3) secure storage and 4) integrity reporting. The most important feature provided by secure boot for resource-constrained RTES is the ability to authenticate all the components that are needed by the platform, starting from the root of trust. The image authentication operation consists of verifying the signature provided in certificates. We assume all images are signed by the developer.

Security Module. A dedicated security module allows to execute the cryptographic instructions (hash computation, signature comparison). Security modules usually come in the form of a trusted platform module (TPM) [4] or a hardware security module (HSM) [5]. TPMs are embedded on the platform while HSMs are external (removable) and are generally plugged into the system using a system port or via a TCP/IP connection. Although both options require that the data used by the module be securely transferred to and from it, a networked-connected system might be vulnerable the connection between the HSM and the safety unit represents an attack surface: an adversary can exploit this connection to gain access to the otherwise isolated safety unit of the system and compromise the safety controller, the restart mechanism, as well as the decision unit. Therefore, although a TPM costs more in design area, we choose this option [1]. Also, TPMs generally offer some storage in the form of nonvolatile random access memory (NVRAM) for keys and sensitive data which will be sufficient for our purpose.

The proposed mechanism is illustrated in Fig. 2. The arrows indicate the order in which events occur in the secure boot sequence. At the beginning, a reboot signal is sent to the TPM which acts as the root of trust as shown in ①. The TPM verifies the integrity of the bootloader (see marker ②). After a successful verification, the TPM provides the security services requested by each component added to the trust chain, starting with the bootloader. Before a component is added to the chain, its integrity is measured (indicated by arrows ③ and ⑥). If the measurements obtained by the authenticating component match the expected value stored in the NVRAM, the component is executed (④ and ⑦). Then, the platform configuration register of the TPM containing the current hash measurement of the current chain of trust is extended by integrating the hash of the new component (⑤ and ⑧).

B. Impact on Schedulability and System Availability

The restart-based approach to security relies on the fact that the booting sequence does not disturb the normal operation of the plant. In Section IV, we show that by introducing a secure boot mechanism, the restart process is lengthened. In the following, we estimate this delay for single and multiple restarts.

Single-Restart Disturbances. We consider that there has been no disturbance to the system due to a particular restart task $\tau_{r,k}$ if, when restart is triggered, the schedule can accommodate *all* the following activities without missing any deadline: **t1**: execution of the restart sequence; **t2**: execution of all ready tasks that have been suspended due to the restart

[1]However, this does not preclude the use of HSM.

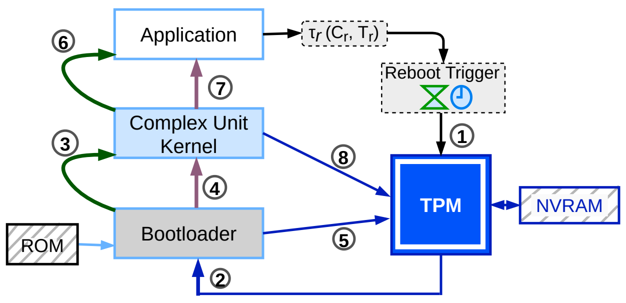


Fig. 2. Secure Boot Sequence for Restart-Based Systems. The Chain of trust starts with the TPM root of trust, followed by the bootloader, the Complex Unit kernel and applications. Green and purple arrows indicate the verification and execution of a new component respectively. Blue arrows indicate an update of the platform configuration register to include the new component of the chain. Circled numbers indicate the order of operations.

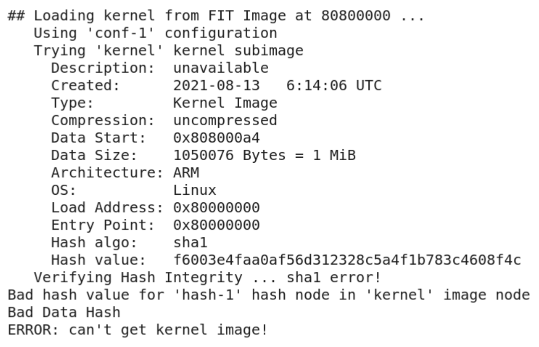
(this includes all tasks that were preempted prior to restart and require a repeat execution).

The duration of **t1** solely depends on C_r, the execution time of $\tau_{r,k}$. In Section IV, we show that by introducing a secure boot mechanism, the restart process is lengthened. That is, with the secure boot enabled, the new worst case execution time of $\tau_{r,k}$ can be expressed as $C_r^* = C_r + \epsilon$, where ϵ is the maximum delay introduced.

On the one hand, **t2** depends on tasks that have not started executing prior to restart, and on the other, on the number of tasks that have started (but not completed) execution. The first group does not introduce any delay due to the restart. For the second group of tasks, we can estimate that in the worst case scenario, there is *one instance $\tau_{i,k}$ of every task in the system that has preempted prior to restart and must be re-executed*. A repeated execution of $\tau_{i,k}$ means that the system must spend an additional time μ_i to reprocess the initial portion of the task that has already been executed before the preemption. Thus, we can formulate the maximum delay related to repeated executions as $\sum_{i=1}^{n} \mu_i$. Therefore, the (total) worst delay δ_s for a single restart is:

$$\delta_s = \epsilon + \sum_{i=1}^{n} \mu_i, 0 < \mu_i < C_i \tag{1}$$

Disturbances for Multiple Restarts. Equation 1 provides the delay associated with one restart sequence. In this section, we extend our analysis to estimate the delay over an entire hyperperiod. In order to do so, we must compute the number N of restart operations that occur during the hyperperiod. Recall that T_r represents the minimum amount of time between two restarts. Thus, the maximum number of restarts during a hyperperiod is $\lceil h/T_r \rceil$. Recall that we have three methods (see Figure 1) to trigger a system restart (watchdog timer, monitoring unit, periodic timer). We can estimate that the minimum number of restarts occurs when *only the periodic timer has been triggered during the hyperperiod*. Therefore, we express the minimum value of N as $\lfloor h/T_{PTimer} \rfloor$, where T_{PTimer} is the periodicity of the periodic timer (which can be obtained from existing work [6]). Having estimated the bounds



```
## Loading kernel from FIT Image at 80800000 ...
   Using 'conf-1' configuration
   Trying 'kernel' kernel subimage
     Description:  unavailable
     Created:      2021-08-13   6:14:06 UTC
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x808000a4
     Data Size:    1050076 Bytes = 1 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x80000000
     Entry Point:  0x80000000
     Hash algo:    sha1
     Hash value:   f6003e4faa0af56d312328c5a4f1b783c4608f4c
   Verifying Hash Integrity ... sha1 error!
Bad hash value for 'hash-1' hash node in 'kernel' image node
Bad Data Hash
ERROR: can't get kernel image!
```

Fig. 3. Secure boot prevents booting a compromised kernel in the complex controller.

of N, we can express the delay $\delta_h = N\delta_s$ over a hyperperiod as:

$$\left\lfloor \frac{h(\epsilon + \sum_{i=1}^{n} \mu_i)}{T_{PTimer}} \right\rfloor \leq \delta_h \leq \left\lceil \frac{h(\epsilon + \sum_{i=1}^{n} \mu_i)}{T_r} \right\rceil \tag{2}$$

IV. Feasibility of the Proposed Design

We conducted two experiments to analyze the security and overhead of the secure boot in a simplex-based CPS system. For our experimentation, we have used an ARM-based Beaglebone Black (BBB) System on Chip (SoC) and the Real-Time Executive for Multiprocessor systems (RTEMS) [7] as our real-time kernel. RTEMS is a POSIX compliant hard real-time OS with high temporal predictability. The Python-based tool `grabserial` is used to collect timestamps for each line of the console output to find out the total restart time for the board. We used the *verified boot* feature of the *Das U-Boot* bootloader on our test platform and signed a kernel image with a RSA-2048 key [8]. The bootloader verifies the integrity of the kernel with the SHA-1 algorithm.

Experiment 1 - Enhanced Security with Secure Boot: To test the security provided by the secure boot, we modified one byte in the kernel image and placed the modified kernel image in the SD card of the board. After this modification, we proceeded to restart the system. As shown in Fig. 3, our protection technique prevents booting the manipulated image. This shows that the integration of secure boot can enhance the security of restart-based systems.

Experiment 2 - Secure Boot Delay. In this experiment we compare the boot times of our proposed secure boot proof-of-concept implementation with the current (unmodified) bootloader of RTEMS. From 1000 runs we observe that the boot time overhead is about 28.56 ms on average (see Fig. 4).

V. Discussion

The time overhead added by the secure boot can have an impact on both the system's availability and performance. However, most restart-based systems face a similar challenge since the traditional restart always introduces a certain amount of delay. On the positive side, our approach enhances the security of the system from some attacks.

Our solution uses the secure boot and chain of trust to prevent attackers from getting a persistent foothold on the

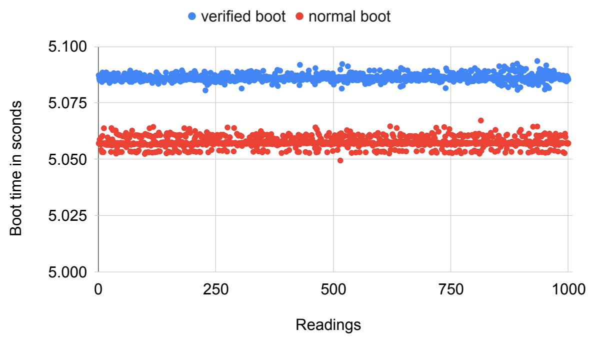


Fig. 4. Reboot time comparison of secure boot vs non-secure boot restart sequence.

TABLE I
COMPARISONS BETWEEN THE STATE-OF-THE-ART AND OUR APPROACH, FT=FAULT-TOLERANT, SG=SAFETY-GUARANTEED, RD=RANDOMNESS, SB=SECURE BOOT.

Full-System Boot Assurance Existing Architecture	Features			
	FT	SG	RD	SB
Simplex-based [9]–[12]	✓			
System-level Simplex-based [13]	✓	✓		
Revival-based Restart [14], [15]	✓	✓		
Rejuvenation-based Restart [6], [16]	✓	✓		
Randomized state restart [17], [18]			✓	
Our Approach	✓	✓		✓

complex partition. While building the chain, any component that fails the verification will cause the system to lock. The proposed design can serve as an *"indicator of compromise" (IOC)* and greatly burden the adversary by making it more difficult to gain a persistent foothold in the system.

VI. RELATED WORK

Simplex-based assurance architecture [9]–[12] focuses on fault-tolerant and low-overhead solutions, while not necessarily guaranteeing safety. System-level Simplex [13], and restart-based (both revival [14], [15] and rejuvenation [6], [16]) add a safety guarantee to Simplex-based architecture. Diversification-based security [17], [18] is a mechanism that introduces execution path randomness after every restart by diversifying configuration files, memory location, and hardware state to reduce exposure of system vulnerabilities. In contrast, we propose enhanced security in the system architecture by adding a layer of secure boot in the CPS. These works are summarized in Table I. Our approach seeks to complement all fault-tolerant, low-overhead, safety-guaranteed, and randomized approaches for robust, secure and safe cyber physical systems.

VII. CONCLUSION AND FUTURE WORK

In this work, we have presented a secure boot mechanism for restart-based real-time CPS by leveraging the isolation provided by the Simplex architecture. We analyzed the proposed architecture based on the security and runtime overhead. Our theoretical analysis on the potential disturbance (caused to the CPS due to multiple restarts) paves the way for a more detailed study on the performance overhead and its impact on schedulability, which is a promising direction for future work. Our future efforts will focus on optimizing this delay by identifying bottlenecks in the security mechanism. We intend to increase the robustness of this security feature by relaxing some assumptions such as the isolation of the safety partition. We are also working on improving the IOC-metric by introducing an *entropy extraction framework* to meet intrusion detection system requirements for resource-constrained subsystems in CPS.

REFERENCES

[1] L. Sha, R. Rajkumar, and M. Gagliardi, "Evolving dependable real-time systems," in *1996 IEEE AeroConf.*, vol. 1. IEEE, 1996, pp. 335–346.
[2] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE RTAS*. IEEE, 2009, pp. 99–107.
[3] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Restart-based security mechanisms for safety-critical embedded systems," *arXiv preprint arXiv:1705.01520*, 2017.
[4] H. Brandl, "Trusted computing: The tcg trusted platform module specification," *Infineon Technologies AG*, 2004.
[5] S. Smith, "Hardware security modules," in *Handbook of Financial Cryptography and Sec.* Chapman and Hall/CRC, 2010, pp. 283–304.
[6] F. Abdi, M. Hasan, S. Mohan, D. Agarwal, and M. Caccamo, "Resecure: A restart-based security protocol for tightly actuated hard real-time systems," *IEEE CERTS*, pp. 47–54, 2016.
[7] G. Bloom, J. Sherrill, T. Hu, and I. C. Bertolotti, *Real-Time Systems Development with RTEMS and Multicore Processors*. CRC Press, Nov. 2020.
[8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[9] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.
[10] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*. IEEE, 2011, pp. 3–12.
[11] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," in *2014 IEEE RTSS*. IEEE, 2014, pp. 138–148.
[12] F. Abdi, R. Tabish, M. Rungger, M. Zamani, and M. Caccamo, "Application and system-level software fault tolerance through full system restarts," in *2017 ACM/IEEE 8th ICCPS*. IEEE, 2017, pp. 197–206.
[13] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE RTAS*. IEEE, 2009, pp. 99–107.
[14] F. A. T. Abad, R. Mancuso, S. Bak, O. Dantsker, and M. Caccamo, "Reset-based recovery for real-time cyber-physical systems with temporal safety constraints," in *2016 IEEE 21st ETFA*. IEEE, 2016, pp. 1–8.
[15] P. Jagtap, F. Abdi, M. Rungger, M. Zamani, and M. Caccamo, "Software fault tolerance for cyber-physical systems via full system restart," *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 4, pp. 1–20, 2020.
[16] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *2018 ACM/IEEE 9th ICCPS*, pp. 10–21.
[17] M. Arroyo, H. Kobayashi, S. Sethumadhavan, and J. Yang, "Fired: frequent inertial resets with diversification for emerging commodity cyber-physical systems," *arXiv preprint arXiv:1702.06595*, 2017.
[18] M. A. Arroyo, M. T. I. Ziad, H. Kobayashi, J. Yang, and S. Sethumadhavan, "Yolo: frequently resetting cyber-physical systems for security," in *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2019*, vol. 11009. International Society for Optics and Photonics, 2019, p. 110090P.