Secure Reboots for Real-Time Cyber-Physical Systems

Vijay Banerjee
vbanerje@uccs.edu
University of Colorado Colorado Springs
Colorado, USA

Sena Hounsinou
shoueto@uccs.edu
University of Colorado Colorado Springs
Colorado, USA

Habeeb Olufowobi
habeeb.olufowobi@uta.edu
University of Texas Arlington
Texas, USA

Monowar Hasan
monowar.hasan@wichita.edu
Wichita State University
Kansas, USA

Gedare Bloom
gbloom@uccs.edu
University of Colorado Colorado Springs
Colorado, USA

ABSTRACT

Cyber-Physical Systems (CPS) such as industrial control systems, automobiles, and medical devices often consist of applications with real-time properties. Due to the safety-critical nature of the application domain, multiple security and fault tolerance approaches have been studied and used in safety-critical CPS. One of the popular approaches for CPS safety is the Simplex architecture, which has also been used recently to strengthen the security of the CPS. The simplex architecture supports the integration of safe controllers for dependable systems, and when combined with periodic restarts, the architecture can reset the CPS into a safe state after each restart. However, these restart-based systems do not protect the system against attacks that persist beyond a restart. Such attacks can be mitigated using secure boot, which is a widely used approach for securing general computing systems but is not used in real-time systems due to the overhead of the boot process. This paper presents an analytical framework and derives feasibility conditions to enable secure reboots in real-time applications. The schedulability conditions presented can be used to design and integrate secure reboot into Simplex-based CPS. Our analysis shows that secure boot adds a deterministic and low-performance overhead, which can be as low as 0.08%.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Real-time systems**; • **Security and privacy** → **Trusted computing**.

KEYWORDS

Secure Boot, Real-Time Systems, Cyber-Physical Systems

ACM Reference Format:
Vijay Banerjee, Sena Hounsinou, Habeeb Olufowobi, Monowar Hasan, and Gedare Bloom. 2022. Secure Reboots for Real-Time Cyber-Physical Systems. In *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy (CPSIoTSec '22), November 7, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3560826.3563384

CPSIoTSec '22, November 7, 2022, Los Angeles, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9876-3/22/11.
https://doi.org/10.1145/3560826.3563384

1 INTRODUCTION

The application of cyber-physical systems (CPS) has increased tremendously across a wide range of domains, including safety-critical applications such as flight software, medical devices, and scientific instruments. CPSs have also taken advantage of the recent improvements in the Internet of Things (IoT) technologies to streamline processes and enhance performance and productivity. However, these improvements require an increased level of connectivity for many components in the CPS architecture, including the real-time embedded systems (RTES), which CPS rely on to control certain core processes of the plants. As a result, RTESs have become more exposed to adversaries that may compromise the controller in an effort to affect the plant.

Increasing attacks on CPS have motivated research in CPS security. One such security approach is the Simplex architecture. Figure 1 depicts the architecture comprising three primary components: safety unit, complex unit, and decision module. The safety unit contains a *fully verified* controller that is outside the reach of an attacker. The complex controller, on the other hand, makes significant use of commercial off-the-shelf (COTS) components that are not verified and can be vulnerable to attacks. The decision module is responsible for switching the operation mode between safety and complex controller to ensure the CPS plant is functional throughout the timeline. The use of the COTS component makes the complex controller exposed to known vulnerabilities. Restart-based approach has been previously used to strengthen the security of the complex controller [1–3].

In this work, we present a secure boot integrated restart-based approach that periodically restores the real-time complex controller into a secure computing environment. The secure boot mechanism prevents the installation of persistent rootkits or compromised OS images from taking over a system. Though the secure boot sequence ensures a trusted computing environment after every restart, its use in safety-critical systems is limited due to the lack of thorough timing analysis that is needed for real-time systems.

This paper presents a response-time based analysis of periodic secure reboots and establishes a relation between the schedulability

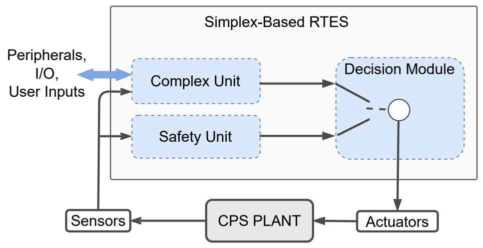


Figure 1: Architecture of CPS plant with Simplex-based RTES Controller.

and vulnerability window as a function of the system parameters like reboot period, system unavailability, and scheduling scheme.

Our contributions are summarized as follows:

- We introduce a restart-based real-time security framework based on periodic secure reboots.
- We provide schedulability analysis and equations that can be used to derive performance trade-offs arising from the level of security requirements.
- Using the system's task parameters, we characterize the delay introduced by the non-instantaneous periodic secure restart and determine its bounds.

2 BACKGROUND

A system with real-time properties requires that the application tasks complete execution before a predetermined *deadline*. This timing guarantee is crucial, especially for safety-critical CPSs, to avoid system failure, computed by *schedulability analysis* that verifies whether *all* tasks in the system will meet their deadlines in the worst-case. A method for establishing the worst-case scenario is through a characterization of the worst-case response time (WCRT) [4, 5] computed by adding the worst-case execution time (WCET) of a task and the interference caused due to preemption by higher priority tasks. The WCET can be calculated using various methods such as a tree- and path-based analysis [6, 7].

Secure boot establishes a trusted computing environment in a system after each restart by ensuring that a device only boots from trusted software (firmware). This trust is accomplished by authenticating each component of the boot process, starting with a (hardware or software) root of trust component. The authentication consists of verifying the embedded signatures contained in firmware images to determine whether to allow or deny its execution, thereby preventing the installation of persistent rootkits or compromised OS images from taking over a system. Compared to a normal reboot (i.e., a reboot with no signature verification) [7], the secure boot authentication process requires more time, and integrating a recurring secure reboot to maintain a trusted environment in real-time CPSs will impact system availability. Specifically, periodically rebooting an RTES controller can have additional consequences for the CPS's operation since the plant's actuators rely on timely commands generated by the RTES.

3 ADVERSARY MODEL

As shown in Figure 1, the complex controller unit of the RTES can be accessed from the CPS network through the system's input interfaces and peripherals. In this work, we consider a remote adversary that can reach the RTES through these interfaces and compromise the software of the complex controller. We assume that the safety controller is located on a separate and isolated partition of the RTES and therefore cannot be accessed by the remote attacker. We also assume that all software components of the complex unit in the RTES are trustworthy initially, i.e., their static images are cryptographically signed by trusted system developers.

Consistent with previous studies [8], the goal of the attacker in this work is to control or tamper with the plant's operation. To achieve this goal, the attacker manipulates the real-time operating system (RTOS) image or real-time and control applications in the complex controller. This manipulation impacts the integrity of the actuator commands computed by the complex controller.

4 SECURE BOOT-ENABLED SIMPLEX SYSTEM

Our goal in this work is to integrate security functionalities to prevent the adversarial actions presented in Section 3 while attempting to minimize the impact on the system's performance. In this section, we present the design of the secure boot-enabled RTES and analyze its schedulability performance.

4.1 System Design

An RTES compromised by an adversary at runtime can be recovered by resetting the complex partition to its initial trusted state using an external timer input and a secure boot-enabled restart operation. The safety unit provides a mechanism to initiate the secure restart operation on the complex unit, and regardless of the current state of the partition (secure or under attack), the safety unit sends a hardware pulse to the complex controller reboot pin. Hence, an adversary is unable to block the restart operation. The secure-boot enabled restart ensures that (1) compromised software components are disabled, and (2) only authenticated software components are activated in the complex unit once the partition is restored.

Both the deactivation (of possibly corrupted software) and authentication (of trustworthy software) are achieved through a signature verification mechanism started from the root of trust, a system component trusted for measurement and verification at all times. The root of trust comes both in the form of software and hardware components. A software root of trust is typically stored in a secure read-only memory (ROM) location and is responsible for checking the signatures of subsequent components locally or with the help of trusted hardware. In this work, we use a bootloader as a software root of trust, a trusted component in traditional computing systems that initializes a system's software stack. A bootloader is also usually lightweight and would be suitable for resource-constrained platforms such as an RTES. In addition, it is readily available on most COTS devices or would be easy to install on an existing system that does not have one.

This signature verification approach guarantees a secure computing environment *only at restart*. That is, once the RTES has been securely rebooted, the adversary can once again attempt to modify the RTOS and other applications to regain control of the complex controller. Thus, to limit the potential impact of an attack, we routinely verify the authenticity of the software on the platform by performing a periodic secure reboot.

4.2 Schedulability Analysis

We now analyze the operation of the complex unit with periodic secure reboots enabled and derive the schedulability conditions for a secure-reboot system. First, we formally define the RTES tasks necessary for the analysis as follows: we consider that the complex controller is a uniprocessor system that executes a set $\mathcal{T}$ of periodic tasks using a fixed priority preemptive scheduling algorithm prior to integrating the periodic secure reboot functionality. Each task $\tau_i \in \mathcal{T}$ is characterized by a tuple $\{C_i, T_i, \pi_i\}$, where C_i is the WCET of the task. T_i is the period i.e., an instance of the task is periodically released at a regular interval of T_i units of time (we denote by $\tau_{i,m}$ the instance released at time mT_i). π_i is the task's priority. Priorities are assigned such that for two tasks τ_i and τ_j, if $\pi_i < \pi_j$ then τ_i has higher priority than τ_j. In addition, we assume the tasks to have *implicit deadlines*, that is, a task released at mT_i, for an arbitrary integer m, must complete its execution by $(m+1)T_i$ (we refer to the implicit deadline simply as deadline herein). We denote the total utilization of $\mathcal{T}$ by $U = \sum u_i$ where $u_i = C_i/T_i$.

The WCRT of τ_i on a unicore processor, with fixed priority preemptive scheduling [9, 10], can be calculated using the recurrence relation by Audsley et al. [4]:

$$R_i(n+1) = C_i + \sum_{\pi_j < \pi_i} \left\lceil \frac{R_i(n)}{T_j} \right\rceil C_j \tag{1}$$

where $R_i(n)$ is the value of the WCRT calculated at nth step of the iteration. The equation terminates when $R_i(n+1) = R_i(n)$ or $R_i(n) > T_i$. The base condition for the recurrence relation can be taken as $R(0) = C_i$.

To integrate the periodic secure reboot, we model the reboot procedure as a periodic task τ_r with a WCET of C_r, a period of T_r and a priority π_r. Since the reboot process is capable of preempting all ongoing processes in the complex controller, we consider that τ_r has the maximal priority π_r in the system, i.e., $\pi_r < \pi_i$, $\forall \tau_i \in \mathcal{T}$. Also, for the restart task, the WCET C_r can be viewed as the duration between triggering the reset pin of the controller to the instant the first task of $\mathcal{T}$ starts execution. This duration depends on the controller's mode of operation. For the purpose of our analysis, we distinguish three modes: for a system with no-restart task, we let $C_r = 0$. When a periodic non-secure restart is added, we can assign $C_r = \epsilon$, where ϵ represents the duration of the system restart. Finally, for a mode of operation that integrates the periodic secure reboot functionality, $C_r = \epsilon + \epsilon'$, where ϵ' represents the overhead due to secure boot verifications. Since the restart procedure is the same for every restart, C_r is considered to be constant.

Besides C_r, another timing parameter that we must study in order to perform an accurate schedulability analysis is the maximum number of restarts that a task τ_i can be subject to before completing a single execution. The worst-case number of restarts can be characterized by first understanding the secure reboot mechanism: if a task is already executing while the reboot is triggered, that task will be terminated and flushed along with the rest of the system memory. However, if the task has not been released yet, that task will be scheduled even if the task and restart are released at the same time. Using this distinction, we formulate the following Lemma derived from Eq. 1:

Lemma 4.1. *WCRT for an arbitrary task in a secure-restart based RTES is found when*

$$R_i(n+1) = C_i + C_r + \sum_{\pi_j < \pi_i} \left\lceil \frac{R_i(n)}{T_j} \right\rceil C_j \tag{2}$$

converges, i.e., $R_i = R_i(n+1) = R_i(n)$.

Proof. Let us draw a relation between an arbitrary task $\tau_i \in \mathcal{T}$ and the reboot task, τ_r for an arbitrary instance $\tau_{i,m}$. We know that the release time of $\tau_{i,m}$ is mT_i and the deadline is $(m+1)T_i$. Similarly, we can assume a reboot task with period T_r. The least common multiple of the periods of all tasks except the reboot task is termed as a *hyperperiod*, which can be denoted by h. The total number of possible restart in h can be calculated as $\lfloor \frac{h}{T_r} \rfloor$. Out of all the possible reboot instances, let us assume that instance $\tau_{r,k}$ is closest to $\tau_{i,m}$. Between $\tau_{i,m}$ and $\tau_{r,k}$, there can be three possible relations: $kT_r \le mT_i$, $mT_i < kT_r \le (m+1)T_i$, and $kT_r > (m+1)T_i$.

Case 1: The worst-case overhead will occur at $kT_r = mT_i$, where the overhead will be $C_r = \epsilon + \epsilon'$.

Case 2: In this case, either $kT_r - mT_i$ is large enough for $\tau_{i,m}$ to complete the execution, or the reboot will terminate the task and it will be deemed non-schedulable. Hence, the worst-case overhead will be 0 in this case.

Case 3: This case won't affect the execution of $\tau_{i,m}$. Hence, this case won't add an overhead to the WCRT.

Hence, the highest interference due to reboot will be observed in case 1. Therefore, taking $C_r = \epsilon + \epsilon'$ in Eq. 2 captures the WCRT out of all possible cases. □

The key here is that a task instance can face a maximum of one reboot during its execution. If the reboot preempts the task, the decision unit will switch the control to the safety unit to prevent the system from crashing and will switch back to the complex controller when it is active again after the reboot.

A task is deemed non-schedulable if any instance of the task fails to complete execution within the deadline. Lemma 4.1 states the WCRT equation for a periodic reboot. The WCRT for a task can be used to formally state the conditions for schedulability.

Lemma 4.2. *For a task τ_i to be schedulable in a secure reboot enabled RTES, it is necessary to satisfy the following conditions:*

(1) $R_i \le T_i$,
(2) $U + u_r \le 1$, *where u_r is the utilization of τ_r,*
(3) $R_i \le T_r$,

If condition 1 is not satisfied, a task released at time mT_i will not be able to complete execution before the release of the next instance of the task time $(m+1)T_i$. Condition 2 can be explained by the following reasoning: Let $U = \frac{C}{T}$, and $u_r = \frac{C_r}{T_r}$. If $\frac{C}{T} + \frac{C_r}{T_r} > 1$, it implies that $\frac{CT_r + C_rT}{TT_r} > 1$. We know that the hyperperiod of a task set is equal to the LCM of the period of all the tasks. Let us denote a hyperperiod by h. Hence, we can also write $\frac{CT_r + C_rT}{h} > 1$, or $CT_r + C_rT > h$, which implies that all the tasks (including the restart task) cannot be accommodated within the given hyperperiod if condition 2 is not satisfied.

Condition 3 of Lemma 4.2 is an extension to condition 1 and it is only applicable to systems with periodic reboots. The conditions state that the reboot period has to be at least the length of WCRT

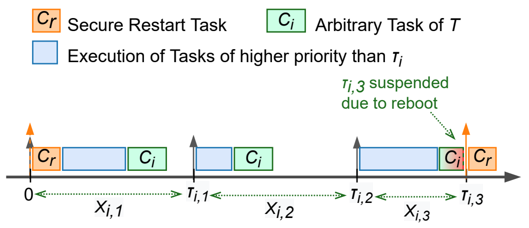

C_r Secure Restart Task
C_i Arbitrary Task of T
Execution of Tasks of higher priority than τ_i
$\tau_{i,3}$ suspended due to reboot
C_r
C_i
C_i
C_i
C_r
0
$\tau_{i,1}$
$\tau_{i,2}$
$\tau_{i,3}$
$X_{i,1}$
$X_{i,2}$
$X_{i,3}$

Figure 2: Example of three execution windows ($X_{i,1}$, $X_{i,2}$, and $X_{i,3}$) for task τ_i. The minimum execution window is $X_{i,3}$ because it is shorter than $X_{i,1}$ and $X_{i,2}$ due to the upcoming second instance of τ_r.

of the task, else the task will be terminated by the periodic reboot and it will never be able to complete execution.

For a periodic task set, we can treat the necessary schedulability conditions stated in Lemma 4.2 as the base condition for schedulability. However, these conditions are not sufficient to prove a task's schedulability because these conditions do not account for all the possible instances of a task. As stated in the proof of Lemma 4.1, there can be cases where $kT_r - mT_i \leq R_i$ and the task will fail to complete before being terminated by the system reboot. We can generalize the cases from the proof of Lemma 4.1 to define the *execution window* of the task.

Definition 4.3 (Execution Window). For an arbitrary instance m of a task, $\tau_{i,m}$, the execution window is the maximum available execution time before the task gets terminated. The execution window of any arbitrary instance $\tau_{i,m}$ can be formally defined as:

$$X_{i,m} = \begin{cases} kT_r - mT_i & mT_i < kT_r \leq (m+1)T_r \\ T_i, & kT_r \leq mT_i \text{ or } kT_r > (m+1)T_i \end{cases} \quad (3)$$

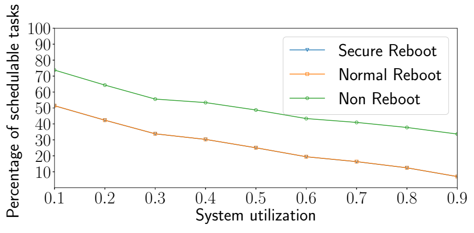
For all possible pairwise value of (m, k), where m and k are instances of a system task and reboot task respectively.

In Fig. 2, we see three different cases of execution window. For $X_{i,1}$, the task executes after the system has rebooted into a fresh state, in this case, the execution window is T_i. In the case of $X_{i,2}$, we see no interference due to reboot and this case also has the same execution window size. However, in the case of $X_{i,3}$ we note that the execution time has been shortened due to the periodic reboot. Using the three cases, Eq. 3 can be further tightened to only use instances of T_r to calculate the execution windows that are shortened due to the periodic reboot. We can derive a definition for the minimum available execution window for an arbitrary task based on the reboot period.

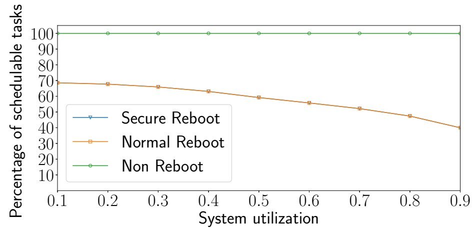
Definition 4.4 (Minimum Execution Window). The shortened execution window can be defined as:

$$X_{i,m} = \begin{cases} kT_r - \left\lfloor \frac{kT_r}{T_i} \right\rfloor T_i, & kT_r \mod T_i \neq 0 \\ T_i, & kT_r \mod T_i = 0 \end{cases} \quad (4)$$

In the piecewise equation Eq. 4, the conditions are based on the divisibility of kT_r by T_i. If the reboot instance is a multiple of T_i, i.e., $kT_r = nT_i$ for $n \in \mathbb{Z}$, the execution window of the immediately preceding task will be $kT_r - \frac{kT_r}{T_i}T_i = nT_i - nT_i = 0$, since $kT_r = nT_i$.

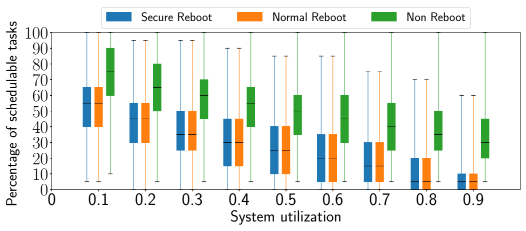


(a) AFPP Scheduling

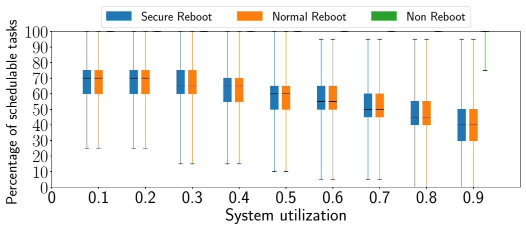


(b) RM Scheduling

Figure 3: Impact of Secure Boot on Task Set Schedulability using AFPP and RM Scheduling.



(a) AFPP Scheduling



(b) RM Scheduling

Figure 4: Impact of Secure Boot on Schedulability.

Hence the execution window available to the *nth* instance of the task is T_i as the reboot trigger coincides with the task deadline and the task window is not shortened by the reboot. Using Eq. 4, the Minimum Execution Window can be defined as $min\{X_{i,m}\}_{m=0}^{N}$, where $N = \frac{h}{T_i}$.

THEOREM 4.5. *If the WCRT of a periodic task τ_i, calculated using Lemma 4.1, satisfies the conditions of Lemma 4.2, then the task is guaranteed to be schedulable if the minimum execution window is at least as long as the WCRT of the task.*

Proof. Let us assume that task τ_i satisfies Lemma 4.2 where the WCRT, R_i, is calculated using Lemma 4.1, which accounts for all the possible worst-case interference. Let us assume that the Minimum Execution Window = X^{min}. If $R_i < X^{min}$, then using Definition 4.4, $R_i < X_{i,m} \forall 0 < m \leq \frac{h}{T_i}$, which says that if the WCRT of a task can be accommodated within the minimum execution window, the task can complete execution in all its instances. Recall that the schedulability of a task is defined as the ability to complete execution before the deadline in all the instances of the task. Hence, Theorem 4.5 sufficiently proves the schedulability of any task τ_i. □

5 EVALUATION

In this section, we evaluate the impact of the addition of the secure reboot task on the task set schedulability.

5.1 Experimental Setup

We implemented our approach on the RTEMS RTOS [11] using *Das U-Boot*, which is a popular open-source bootloader that is compatible with a wide range of embedded devices. There are two steps to implementation; the first stage occurs on a host computer and the second on the RTES. During the first phase, The U-boot bootloader is built using user-provided configurations supplied through an `.its` file. Next, the hashed image is encrypted with RSA2048 and stored in a flattened image tree format. In addition to the flattened image tree, the device tree and generated public key are stored in a read-only memory location on the RTES. The second stage occurs every time the RTES is restarted. U-Boot uses the public key obtained from stage 1 to verify the hash of the kernel image and only allows a signature-verified image to boot. To ensure the integrity of execution, we terminate and discard all the tasks that did not complete execution before the reboot was triggered.

For performance analysis of the proposed model, we used Theorem 4.5 on a synthetic task set that we generated using the UUnifast algorithm [12]. With a constant value for the hyperperiod ($h = 1000$), we randomly selected task periods from the set of the factors of h. For each task τ_i, the WCET is calculated using $C_i = T_i \times U_i$. We varied U from 0.1 to 0.9 in steps of 0.1. For each value of U, we generated 1000 task sets with 20 tasks in each task set. The reboot overhead values, i.e., values for ϵ and ϵ', used in this performance analysis are recorded from our hardware test implementation discussed above.

5.2 Experiments and Results

We performed the following experiments to gain quantitative insights about the performance overheads due to periodic secure reboots. We measured performance in terms of the impact on the system's schedulability in three different modes: (1) **No reboot:** $C_r = 0$ (2) **Non-secure reboot:** $C_r = \epsilon$ (3) **Secure-reboot:** $C_r = \epsilon + \epsilon'$.

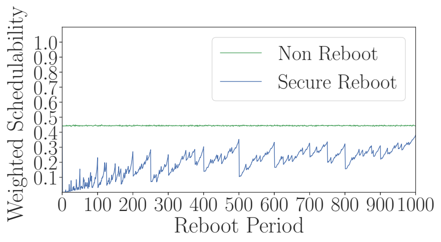
We define *schedulability* of a task set as the percentage of the tasks that can successfully complete execution before their respective deadline. For each experiment, we used an arbitrary fixed priority preemptive (AFPP) scheduling [9] and a rate monotonic (RM) scheduling algorithm [13] for assigning priorities to each task in the task sets.

Experiment 1: This experiment shows the schedulability comparison of a set of 1000 task sets for each value of U with a randomly chosen fixed value of T_r from a range of $(0, h]$. The reboot overheads are $\epsilon = 5.05$ and $\epsilon' = 0.02856$ seconds, which are collected from our implementation setup. Fig. 3 shows the impact of the periodic restart of the complex controller. Interestingly, we observe an almost indistinguishable pattern in the normal reboot and secure reboot traces with both AFPP and RM scheduling (see Figs. 3a and 3b), which implies that for restart-based systems, there is no significant reduction in schedulability by adding the secure boot sequence in the restart operation. In particular, for the utilization range typically used in CPS (0.5 to 0.7), the maximum drop in schedulability is approximately 0.03% for AFPP and 0.081% for RM scheduling. We attribute this to the schedulability conditions presented in subsection 4.2.

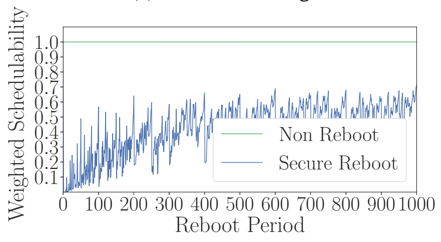
Experiment 2: We extend Experiment 1 further and analyze the numerical summary from a randomly generated task set. To understand the performance trend better, we generated additional task sets and analyzed the schedulability of 50000 randomly generated tasks using boxplots. The five-point summary (minimum, first quartile, median, third quartile, and maximum) schedulability of all tasks shows a complete picture of the task schedulability at the given utilization level with the same fixed values of ϵ and ϵ' as used in Experiment 1. We used a pseudo-random number generator to assign values for T_r. Fig. 4 shows the schedulability of task set with an arbitrary value of $T_r = 120$. The box plot demonstrates the relation between the system utilization and schedulability of task sets with a constant T_r.

Experiment 3: We now examine the impact of weighted schedulability [14] as a function of the utilization level and task schedulability at each utilization level. We define weighted schedulability as: $\frac{\sum_{i=0}^{k-1}(U_i . S(U_i, \epsilon+\epsilon', T_{r,m}))}{\sum_{i=0}^{k-1} U_i}$, where $S(U_i, \epsilon + \epsilon', T_{r,m})$ is the schedulability at utilization level U_i and $T_{r,m}$. The resulting plot in Fig. 5 shows that the schedulability of the task is directly proportional to the reboot period. On one hand, a longer reboot period results in a higher frequency of the periodic reboot, which lowers the task schedulability. On the other hand, using a lower frequency of secure reboot increases the system's vulnerability. We notice a pattern of sudden peak (when the T_r is a factor of h) immediately followed by a steep drop in schedulability. This observation is due to the fact that the task periods are factors of h. The steep drop can be explained by the same argument: when T_r takes values that are immediately after factors of h, every task having a period equal to a factor of h will have an instance released and terminated due to a reboot being triggered, which severely impacts the schedulability of the task sets for those values of T_r. From this experiment and Experiment 1, we infer that the biggest impact on performance comes from the reboot period, and the schedulability can be maximized by setting the reboot period as the least common multiple of a subset of tasks. The subset of tasks can be selected based on factors such as priority, which will guarantee the execution of high-priority tasks. The task subset selection can also be done in such a way that the maximum number of tasks gets executed.

Experiment 4: In Experiments 1 and 2, we compare the schedulability with fixed reboot overhead and fixed reboot period. In Experiment 3, we observe the impact of the reboot period using a

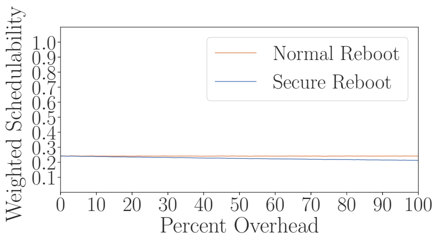


(a) AFPP Scheduling

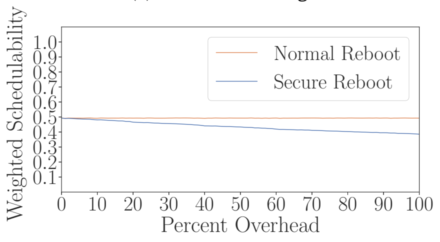


(b) RM Scheduling

Figure 5: Weighted schedulability as a function of reboot period T_r.



(a) AFPP Scheduling



(b) RM Scheduling

Figure 6: Schedulability as function of the secure reboot overhead ϵ'.

weighted schedulability plot with fixed reboot overhead. In this experiment, we demonstrate the impact of the addition of a secure reboot over a normal reboot. Fig. 6 shows a linear depreciation in the schedulability as the overhead percentage of the secure reboot increases. It is interesting to note that while this plot was generated from synthetic task sets, this shows a similar trend when compared to Experiment 1 where we used values from a real system. In Fig. 6a the schedulability difference with $\epsilon' = 0.01 \times \epsilon$ is around 0.03% which is close to what we found in Experiment 1. The weighted schedulability with RM scheduling in Fig. 6b also shows a comparable value 0.1% compared to 0.08% with real system values. Hence, this experiment provides a very significant insight that the impact of secure reboot overhead is low, and decreases linearly.

6 RELATED WORK

Trusted computing aims at protecting systems against integrity attacks by providing an outlet for a root of trust, which uniquely identifies a platform. The Simplex architecture has been used for fault tolerance in control systems utilizing untrusted logic and an isolated safety controller. Approaches based on System-level Simplex architecture [15] and restart-based (both revival [16, 17] and rejuvenation [18, 19]) approaches run the safety controller and decision module on dedicated hardware. These methods add a safety guarantee to the Simplex-based architecture.

Using System-level Simplex architecture, Abdi et al. [20] proposed a restart-based recovery approach for the complex subsystem when software faults are detected. Further, Abdi et al. [8, 18, 19] proposed a framework to periodically restart the platform to improve the safety of real-time systems and provide a system-wide restart-based approach that provides a formal guarantee of system safety. However, these works do not provide proof of timing guarantee and feasibility analysis for real-time systems.

Romagnoli et al. [21] proposed a recovery technique based on software refresh that guarantees the controller integrity and safety. However, recovery does not prevent attacks from occurring again. Simplex-based assurance architecture [2, 22–24] using decision procedures provides fault-tolerant and low-overhead solutions to choose control commands between the complex controller and safety controller to improve system reliability.

Diversification-based security leverages the physical properties of the system to introduce execution path randomness after every restart [25, 26]. Configuration files, memory location, and hardware state are diversified to decrease the exposure of system vulnerabilities after periodic reset operations, which prevents persistent attacks. This work can be combined with our system to provide better security with performance guarantees for real-time systems.

7 CONCLUSION

This paper presents a secure boot mechanism for restart-based real-time CPS leveraging the Simplex architecture. We present a schedulability analysis for the RTES task set when the secure boot is enabled. We evaluated our approach by measuring the impact of periodic restarts with and without the secure boot on schedulability. Experimental results show that the periodic secure boot has a negligible impact when using fixed-priority scheduling schemes. Future work could consider re-execution of terminated tasks, alternative scheduling paradigms that may be amenable to reboot scheduling, and randomization of the reboot timing.

ACKNOWLEDGMENTS

This work is partially supported by NSF CNS-2046705, NSF CNF-2152768, NSF OAC-2001789, and Colorado State Bill 18-086.

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] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[3] 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.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[5] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[6] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[7] S. Hounsinou, V. Banerjee, C. Peng, M. Hasan, and G. Bloom, "Work-in-progress: Enabling secure boot for real-time restart-based cyber-physical systems," in *2021 IEEE Real-Time Systems Symposium (RTSS).* IEEE, 2021, pp. 524–527.

[8] 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.

[9] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *[1990] Proceedings 11th Real-Time Systems Symposium.* IEEE, 1990, pp. 201–209.

[10] M. Gonzalez, H. Mark, H. Klein, and J. P. Lehoczky, "Fixed priority scheduling of periodic tasks with varying execution priority," in *In Proceedings, IEEE Real-Time Systems Symposium.* Citeseer, 1991.

[11] G. Bloom, J. Sherrill, T. Hu, and I. C. Bertolotti, *Real-Time Systems Development with RTEMS and Multicore Processors.* CRC Press, Nov. 2020.

[12] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1, pp. 129–154, 2005.

[13] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, 1989, pp. 166–171.

[14] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of OSPERT*, vol. 10, pp. 33–44, 2010.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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 International Conference on Cyber-Physical Systems (ICCPS).* IEEE, 2018, pp. 10–21.

[20] 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*, 2016, pp. 1–8.

[21] R. Romagnoli, B. H. Krogh, and B. Sinopoli, "Design of software rejuvenation for cps security using invariant sets," in *2019 American Control Conference (ACC).* IEEE, 2019, pp. 3740–3745.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.