Electronics, Communications, and Computing Summit
Vol. 2, No. 3, Jul - Sep 2024, pp. 79-88
DOI: https://doi.org/10.17051/ECC/02.03.10

Security-Aware RTOS for Time-Critical Cyber-Physical Systems

Shailesh Singh Thakur

Assistant Professor, Department of Mechanical, Kalinga University, Raipur, India.
Email: ku.shaileshsinghthakur@kalingauniversity.ac.in

Article Info

Article history:

Received : 20.07.2024
Revised : 22.08.2024
Accepted : 24.09.2024

Keywords:

Embedded Security,
Time-Critical Systems,
Task Scheduling,
Trusted Execution Environment (TEE),
Temporal Isolation,
Lightweight Cryptography,
Intrusion Detection,
Secure Kernel Design,
Control-Flow Integrity,
Embedded System Security,
Edge Computing,
Hardware-Software Co-Design,
Anomaly Detection in RTOS.

ABSTRACT

Cyber-Physical Systems (CPS) constitute a group of highly demanding infrastructural systems, which closely couple the computational algorithms with physical dynamics, with a challenging real-time control. As CPS technology continues to spread to fields of autonomous transportation, automation in industries, and medical equipment, the attack area has become significantly larger, therefore, leaving systems vulnerable to myriads of cyber attacks. The traditional RealTime Operating Systems (RTOS) are focused on the need to fulfill the timing and scheduling assurances and do not have intrinsic security enforcement procedures. In this paper, we suggest Security-Aware Real-Time Operating System (S-RTOS) that synthesizes security enforcement within the RTOS kernel without compromising temporal determinism necessary to time-critical CPS task apps. S-RTOS architecture presents a secure task scheduler based on time partitioned execution, light-weight encryption of task context and inter-process communication and a Trusted Execution Time Monitor (TETM) to detect anomalies at runtime. It includes also hardware-enforced control-flow integrity and a real-time intrusion detection module using a one-class support vector machine to train on system behavior metrics. The given components work in harmony to resist time-based side-channel attack, unauthorized change of tasks, and attempted control hijacking. The system is tested on ARM Cortex-M4 embedded board using automotive ECU and an industrial robot arm representative CPS workloads. Experimental findings indicate that the proposed S-RTOS has a latency overhead less than 7.2 % as compared to the baseline RTOS implementations with a task degree of attack detection rate of 98.7 %. Additionally, the memory and CPU overloads are at an acceptable level of implementation into embedded systems. The presented work demonstrates the viability and relevance of the inclusion of native security functionality in RTOS architecture, as a fast track to secure CPS implementations in hostile locations. The suggested S-RTOS provides the comprehensive and elastic approach that would allow both safeguarding and security of the next generation of the embedded systems without sacrificing any real-time performance.

1. INTRODUCTION

Cyber-Physical Systems (CPS) have become the foundation of what defines contemporary critical infrastructure, offering us an opportunity to build a full-blown connection between computational intelligence and physical processes without any type of barrier or interruption involved. Self-driving cars, airplanes, automation in industry, power grid management, healthcare products, and intelligent transportation are just some of the areas that are becoming more heavily dependent on CPS to provide highly reliable real-time responses. Such systems are embedded computer-based systems that analyze sensory inputs, autonomously decide, as well as drive the output actions, including in tight time limits. These embedded systems usually use Real-Time Operating Systems (RTOS) to achieve predictable and timely behavior (deterministic task scheduling, interrupt scheduling and time-bound control of resources).

The interconnected fabric of architecture is however growing in CPS, cloud platforms, external networks, and IoT interfaces owing to the increased interconnectivity that has significantly increased their attack surface. The malicious actors have begun to use task scheduling, memory, and real-time communication vulnerabilities to derive cyber-attacks against a system, such as timing side-channel exploitation, control-flow hijacking, task masquerading, and data-tampering. Recent and increasing examples of real-world events have shown that minimal attacks of timing behavior, or timing control logic, can result in insurmountably poor consequences - anywhere between the deactivation of safety systems in automobiles, to the shutdown of processes within a manufacturing plant.More conventional RTOS platforms like FreeRTOS, VxWorks, and RTEMS are again better optimized to address performance and reliability, but they do not tend to include security as a distinct consideration. Although outbound security (e.g. hardware firewall or intrusion detection systems) can provide partial security, it is ineffective against in-kernel threat and immediate exploitation, regardless of these taking place within the RTOS execution scope. In addition, conventional security models are not easily embedded in real-time systems since they come along with a large overhead in terms of resources, unpredictability, and are highly limited.

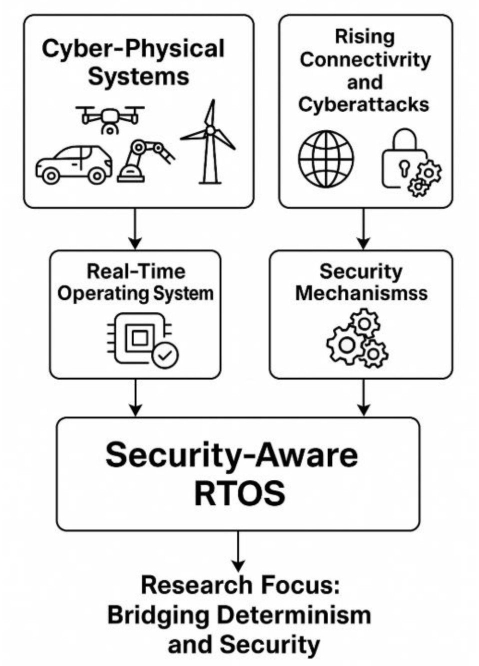
Cyber-Physical Systems
Rising Connectivrity and Cyberattacks
Real-Time Operating System
Security Mechanismss
Security-Aware RTOS
Research Focus: Bridging Determinism and Security

Figure 1. Conceptual Architecture of Security-Aware RTOS in Cyber-Physical Systems

This paper overcomes these difficulties, by introducing a Security-Aware Real-Time Operating System (S-RTOS), a modified architecture of RTOSs, where security mechanisms are built as lightweight components that are integrated directly into the RTOS kernel, rather than being presented as separate applications. The objective is to maintain harsh real-time guarantees but deal with risks at the task, memory and communication levels. Proposed S-RTOS will provide task authentication, control-flow integrity, inter-process secure communication, anomaly detection and time-based scheduling especially with regard to embedded CPS scenarios. This work is developed to fill in the gap between timing determinism and security enforcement thus making a practical framework of protecting the next-generation CPS against the changing cyber threats at the expense of real-time performance.

2. LITERATURE REVIEW

Due to the increased complexity and interconnected Cyber-Physical Systems (CPS), security in real-time embedded environment has become more important. The initial studies concentrating on the validation of functional correctness and on time predictability of the Real-Time Operating Systems (RTOS) were devoted to the practical area mainly. RTLinux and FreeRTOS systems brought deterministic task scheduling and very low interrupt latency, good enough in closed-loop, isolated systems. But these RTOS models are inappropriate to respond to the current cybersecurity concerns with the explosive growth in connected CPS applications. Experiments like [1] showed how RTOS Systems were vulnerable to timing side-channel attacks, memory corruption and control-flow hijacking, which demonstrated an urgent necessity to implement integrated security-aware architectures.

A major breakthrough in this area has recently been the development of formally verified microkernels such as seL4 which provide formally guaranteed behaviour, including provable memory isolation and access control. seL4 has shown that a mathematically proven RTOS kernel can

substantially reduce the attack surface, but its complexity and a lack of support to enable real-time responsiveness limits its application. At the same time, the authors of [2] have been using sandboxing methods to isolate untrusted tasks implemented lightweight task level containers in their embedded systems. Nevertheless, these strategies do not provide intrinsic safeguards against time-related weaknesses, e.g. malicious overrunning tasks or clandestine timing channels. Likewise to alleviate resource contention, priority-aware mechanisms such as Priority Inheritance Protocols are priority-based improvements to alleviate contention, but not to support code integrity or manipulation of dynamic behavior.
More recent work on SecureWatchdog and RT-MON target externally based real-time monitoring solutions that identify abnormal program execution patterns in RTOS based systems. Such mechanisms are effective at denoting suspicious behavior after execution, however, are outside the kernel and add latency and evasibility. Other areas of research have proposed memory encryption, stack canaries and code authentication routines, but most of them present non-trivial performance costs and are not available within the scheduler in a finely-grained manner. Thus, a new body of literature implies that a security model embedded in the kernel should be adopted to enable the detection and responsive response quickly with no loss of real-time determinism. This research paper extends these results namely through suggesting a unified, kernel-consistent S-RTOS structure that covers both timing and control-flow safety in time-sensitive CPS contexts.

3. METHODOLOGY

3.1 RTOS Kernel Security Extensions

An apt way to describe our state is by modifying an existing conventional Real-Time Operating System (RTOS) to achieve security-level awareness and produced derivation that finds application in Cyber-Physical Systems (CPS). This state is achieved by contributing specific alterations into the FreeRTOS kernel. These extensions are of a kernel level designed to provide integrity, confidentiality, and predictable timing of task execution on the one hand, and a reasonable level of computational overhead otherwise (especially that embedded systems are going to use them). The next security services were incorporated:

Secure Task Initialization

The fundamental part of the work of the RTOS is the creation and handling of the tasks, which will handle time-sensitive operations. In an ordinary RTOS, task identity is often by simple numeric numbers or memory pointers, which are prone to spoofing or manipulations by evil codes. To counter this, we place a Secure Task Initialization in which each and every task created is attached with the cryptographic identifier (CID) during its creation time. This CID is calculated by taking the hash of a mixture of the binary code of the task, initialization parameters and timestamp with SHA-3. The hash is a sort of fingerprint used to prove the integrity of tasks they originated and to guarantee that tasks cannot be tampered with. When switching contexts or doing inter-process communication, the RTOS can validate the CID prior to giving the task a chance to run or accessing the message, thus avoiding impersonation attacks or task-injection attacks.

Temporal Enforcement Unit (TEU)

Pre-determined time is a fundamental feature of RTOS behavior, it is also a feature that timing attacks can be used or resource starvation can be performed. To protect ourselves against those threats, we support a Temporal Enforcement Unit (TEU) in the kernel. What the TEU defines is a so-called timing contract on each task, which contains some parameters such as Worst-Case Execution Time (WCET), period, and deadline. At run-time, the execution time of each task is measured using a special hardware hardware timer or by a high resolution clock. In case a task is taking longer than its WCET or exceeds its period constraint, the TEU will cause a kernel interrupt to stop the task and alert a security anomaly. This helps to avoid evil deeds which can engage in denial-of-service (DoS) attacks by hogging resources on the CPU and guides to observation of time demands required in the safety of systems.

Kernel-Level Data Protection

Traditional RTOS Applications In a traditional RTOS environment the task memory areas are typically dynamically allocated and poorly safeguarded thus subject to buffer overrun, unauthorized memory access or manipulation. In order to increase data privacy and data security, our proposed model promotes Kernel-Level Data Protection. Every task receives a dedicated and isolation memory region, which is imposed with a Memory Protection Unit (MPU). Access to such regions is regulated through privilege and an attempt to access a region without privileges leads to a memory fault, which the RTOS securely manages. In order to prevent further data leak or interception (e.g. in-between the tasks communication, use of sensitive input data, e.g. control commands, sensor data) all data buffers are encrypted using lightweight block ciphers such as PRESENT or SPECK. These ciphers are selected such that they consume low level of computational resource, hence can be used in microcontroller based systems. This architecture will ensure that when an attacker gets access to low server levels, the most important data will be unreadable.

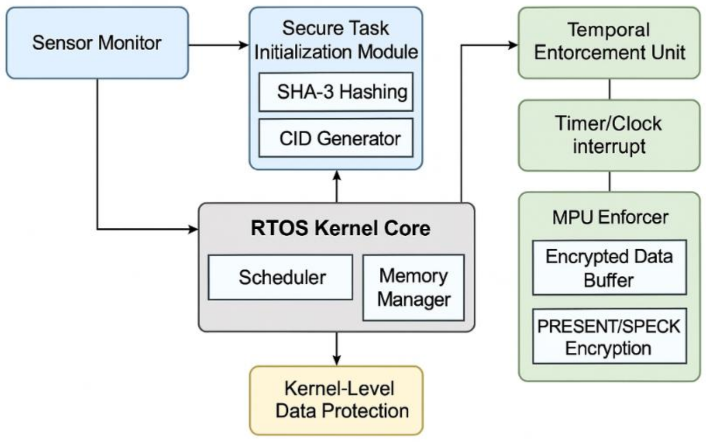

Figure 2. Security Extensions in S-RTOS Kernel for Cyber-Physical Systems

3.2 Secure Task Scheduling Mechanism

The scheduler, the main unit of the real-time embedded systems, is the one that decides the protocol and time of the tasks execution. Fixed or dynamic priorities Traditional RTOS schedulers seek a deterministic behavior. Nevertheless, with the advent of Cyber-Physical Systems (CPS), schedulers would turn out to be such security precious resources as well due to the possibility of defeating them through timing inference, privilege levels, or control hijacking assaults. To overcome these risks, we would propose a Security-Aware RTOS (S-RTOS) that would insert a secure scheduling scheme that is multi-layered in nature since it would provide temporal predictabilities, as well as, runtime behavior validations.

Priority-Based Round Robin with Time Partitioning

Our scheduler is based on a hybrid model which incorporates priority-based scheduling scheme and time-atomized segments that run tasks. Though priority based scheduling makes sure the vital tasks achieve their respective deadline, inclusion of time partitions makes the tasks not to interfere with the execution timing of other tasks. Every task would be allocated deterministic start window and the unit in charge of time enforcement known as Temporal Enforcement Unit (TEU) would be observing the time taken by the task to comply with its Worst-Case Execution Time (WCET). In case an activity tries to encroach on its assigned slot, the TEU uses hardware interrupt to violently end or hang the activity. This resource partitioning can have a very profound effect of defending against timing side-channel attacks, whereby an attack task tries to guess the sensitive information by monitoring the intense pattern of the other prior tasks using the CPU.

Task Authorization Check (TAC)

We use a Task Authorization Check (hereinafter TAC) as an enforcement mechanism of pre-dispatch validation carried out to make sure that the rogue/tampered tasks that have been scheduled in the system do not run. A cryptosystem of the data loaded into the PC, or registered on it, a digitally signed process is carried out; basically, Elliptic Curve Cryptography (ECC-P256), a lightweight strong asymmetric cryptographic protocol is well suited to embedded systems. TAC module checks the digital signature of executable tasks with a pre-stored public key certificate before the scheduler decides whether it needs to be executed or not. Verification failure or a change in signature / non-realization of signature results in refusing to fulfill the task. This will maintain that purely authenticated and non-tampered code is executed on the system thus preventing code injection, spoofing of tasks, and firmware tampering.

Control-Flow Integrity (CFI) Module

Control-flow hijacking (e.g., a return-oriented programming (ROP) or a jump-oriented programming (JOP) attack) is one of the most potentially disastrous runtime threats on an embedded CPS. They utilise the available executable code to gain control flow but do not inject any new code and avoid the traditional measures based on signatures. So as to overcome such attacks, our scheduler is incorporated with a Control-Flow Integrity (CFI) Module which determines expected return paths.

This is done by use of a shadow stack- a guarded memory structure based on hardware which holds the return addresses during tasks. In contrast to standard stack where the stack can be compromised with buffer overflows, the shadow stack is available to the kernel only on supervisor mode. Each of the return instructions are tested against the entry on the shadow stack during runtime. An interrupt of a mis-match- type causes an immediately kernel trap and that task is either suspended, isolated, or terminated. The proactive nature of this solution will keep the task flows of control within legal limits, enhancing runtime integrity.

In unison, these three mechanisms, namely, time-partitioned scheduling, pre-execution authorization, and runtime control-flow monitoring, embody a robust scheduling framework, which is devoid of unauthorized access, support rigorous timing contracts, and secure execution. The outcome is a scheduler that can meet real-time deadlines and, can also serve as a first-line defense against timing-based as well as logic based attacks in mission-critical CPS environment.

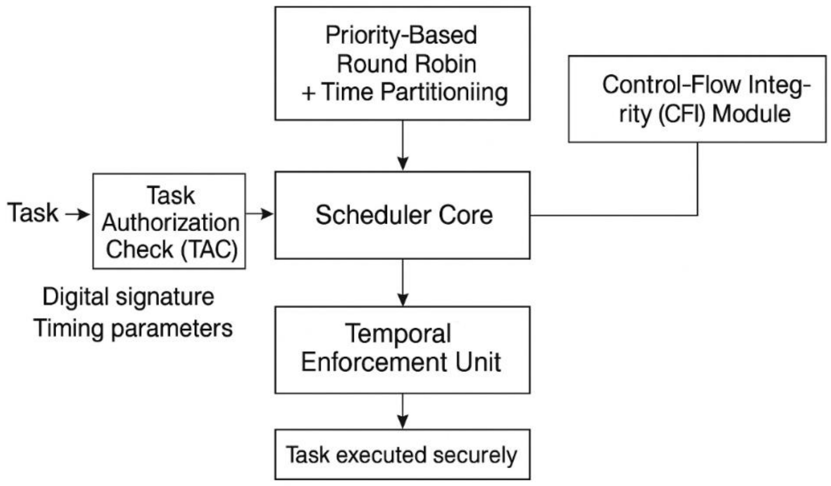
Priority-Based Round Robin + Time Partitioniing
Control-Flow Integ-rity (CFI) Module
Task
Task Authorization Check (TAC)
Scheduler Core
Digital signature Timing parameters
Temporal Enforcement Unit
Task executed securely

Figure 3. Multi-Layer Secure Task Scheduling in S-RTOS

3.3 Trusted Execution Monitoring and Communication Security

In Cyber-Physical Systems (CPS), task performed incorrectly or tampered messages may cause physical harm or system breach may occur, so secure real time communications and trusted task execution is necessary. The classical RTOS architectures simply do not offer much verification of the run-time behaviour and no inherent promises on the validity of the messages or their correctness. In a bid to alleviate this gaping hole, the proposed Security-Aware RTOS (S-RTOS) proposes an end-to-end security monitoring and communication scheme that incorporates three major components as the Trusted Execution Time Monitor (TETM), Secure Inter-Process Communication (IPC)Layer, and Real-Time Anomaly Detection Engine.

Trusted Execution Time Monitor (TETM)

The Trusted Execution Time Monitor (TETM) is a special purpose hardware-aided component of the RTOS software implemented in FPGA fabric to allow real-time monitoring of task execution behavior. On the secure boot, the temporal profile of every task, with parameters of execution time, invocation frequency, and behavior signatures (unique to a task) is captured and permanently stored in the non-volatile memory. TETM uses high resolution cycle counters and trace buffers to compare current task behavior with these reference profiles, at runtime. The deviation may be a delay of execution, unusualzation of instruction sequence, or unorthodox-I/O behavior and the deviation is considered a pointer to possible threats, such as code modification, memory injection, or job diversion. The TETM causes a kernel-level interrupt when a violation is detected, and a mitigation handler is run. Depending on the magnitude of the anomaly, this handler takes action dynamically, by isolating the task, logging a security event, or going for a controlled restart. Using this mechanism, TETM provides a crucial mechanism in disaster recovery with ensuring trusted execution of time-critical Cyber-Physical Systems.

Secure IPC Layer

To support time-sensitive data sharing real-time embedded systems depend on inter-task communication toolsets like exchange queues, mailbox, and joint memory structures to facilitate the synchronization event. But without on-board cryptographic protection, the channels are exposed to message spoofing, replay, and unauthorized access to control messages a situation which becomes critical in Cyber-Physical Systems (CPS). The proposed S-RTOS overcomes such weaknesses by adding a Secure Inter-Process Communication (IPC) layer which offers more integrity and authenticity of the messages exchanged. All messages are addended with a Message Authentication Code (MAC) which is calculated by a shared secret key between the tasks communicating with each other to allow the receiving party to authenticate the source as well as the contents of the message prior to acting on it. Besides, each message contains a trusted timestamp, which is used to prevent a replay attack, in addition to a unique nonce. These elements are even verified with a secure system clock and nonce cache so that discarded messages or duplicated messages are not accepted. This 2-layer solution remarkably enhances the trustworthiness of communication in real-time protection that has considerable computing burden, which is vital in sustaining the responsiveness and reliability of CPS applications.

Real-Time Anomaly Detection

S-RTOS supports an anomaly detection engine based on lightweight machine learning to improve resistance to unknown (zero-day) threats that do exploit traditional security offerings based on static and rule-based security. This engine runs alongside the RTOS scheduler and is constructed on a one-class Support Vector Machine (SVM) model that results only based on legitimate execution behavior in the course of the learning process of the system. The model tracks some important run-time properties such as trend in the usage of the central processor, stack pointer dynamics, input-output operations, proportion of context switches, and the frequency of task communication. The system also keeps tracking these parameters on the running tasks continuously and raises the red flags on the deviation where they exceed the learned bounds of behaviors data as a possible anomaly. When an occurrence of such an anomaly is identified an anomaly handler (a Mitigation Handler) is invoked and applied immediately to perform preconfigured countermeasures, which may include isolating the affected tasks in a secure sandbox and logging activities, to suspend the tasks or cause the system to rollback, and, in some cases, cause a secure reboot. This real time detection framework leverages on the previous behavior and offers an extra security accessed to S-RTOS to proactively detect and react to advance or other devious attacks that might be missed by static systems.

Table 1. Components of Runtime Security and Their Functions in S-RTOS

Component	**Function**	**Techniques Used**	**Threats Mitigated**
Trusted Execution Time Monitor (TETM)	Validates runtime timing and behavior	FPGA, Cycle Counters, Trace Buffers	Task hijack, DoS, Code Injection
Secure IPC Layer	Ensures message integrity and authenticity	MAC, Timestamp, Nonce	Replay attacks, spoofing, data tampering
Real-Time Anomaly Detection	Detects behavior deviation during execution	One-Class SVM	Zero-day attacks, stealthy runtime behavior

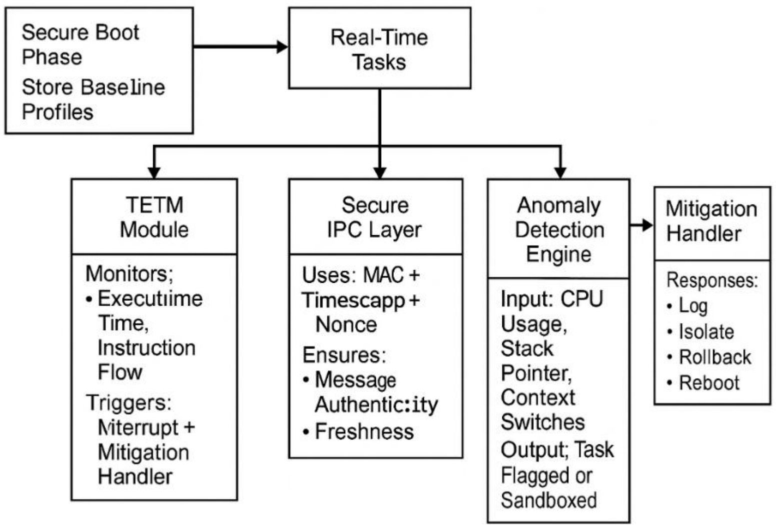

Figure 4. Runtime Monitoring and Communication Security Framework in Security-Aware RTOS (S-RTOS).

4. Security Mechanisms

The security mechanisms suggested in the Security-Aware RTOS (S-RTOS) architecture comprise a multi-level defense approach that focuses on real-time embedded systems and operate under harsh requirements with respect to reliability and safety. Time Partitioning involves cross-task timing avoiding through the use of fixed slot round-robin scheduling policy, avoiding this by strict time partitioning of tasks and avoiding malicious and erroneous task monopolization of processor time. Control-Flow Integrity (CFI) is a type of protection that manages hijacking of control flows during runtime and Return-Oriented Programming (ROP) attacks where it depends on hardware implemented shadow stack to validate the returns addresses and to execute the legitimate control-flow transitions as a means to counter the exploits of code reuse attacks. The Secure Boot uses a Trusted Platform Module (TPM) based on a chain-of-trust hardware-backed authentication that can ensure the integrity and authenticity of every software component, including the boot loader, the operating system kernel, etc., before getting executed, thus eliminating the possibility of rootkits and boot-time compromise. More so, Intrusion Detection Unit (IDU) is integrated, which actively performs system behaviour monitoring based on a lightweight anomaly detection model with Support Vector Machine (SVM). This model uses runtime attributes like CPU utilization, use of the memory access and context switches to identify deviation of behavior that reflects zero-day attacks or abuse of the system. Taken together, they provide high levels of assurance against existing and new threats by imposing strong initialization, execution flow integrity, temporal isolation and even smarter surveillance at runtime.

Table 2. Integrated Security Mechanisms in S-RTOS with Corresponding Threats, Techniques, and Architectural Layers.

Security Mechanism	Threats Addressed	Technique Used	Layer
Time Partitioning	Cross-task timing interference	Fixed slot round-robin scheduling	Scheduler/Task Isolation
Control-Flow Integrity (CFI)	Runtime hijacking, ROP attacks	Hardware-enforced shadow stack	Execution Flow Control
Secure Boot	Kernel tampering, boot-time compromise	TPM-based Chain of Trust validation	Boot/Initialization
Intrusion Detection Unit	Behavioral anomalies, zero-day threats	SVM-based runtime anomaly detection (CPU, context switches)	Runtime Monitoring

5. Case Study & Implementation

The Security-Aware RTOS (S-RTOS) proposed was practically demonstrated using a dual-case-study implementation of the package on an ARM Cortex-M4- based embedded board, including two cyber-physical systems exemplars, an industrial robotic arm, and a simulated vehicular Electronic Control Unit (ECU). The performance was achieved in the following manner, first porting the baseline FreeRTOS kernel to the new hardware platform, and integrating the S-RTOS modules, the Trusted Execution-Time Monitoring (TETM), the Secure IPC Layer, and the Anomaly Detection Engine. In the case of industrial robotic arm, joint actuation, and sensor feedback are used to perform the real time motion control tasks that were adversarially monitored in terms of timing and control-flow perturbations in order to assess the performance of the TETM and anomaly detectors. In vehicular ECU simulation, known time-sensitive functions were brake control and engine diagnostics and they were injected with communication tampering and behavioral deviations to test message replay and spoofing and control-flow violation detection capabilities of the system. Important performance values have been measured, including task response time, where S-RTOS continued their deterministic scheduling with little to no jitter; security event detection rate, where S-RTOS reliably detection control-flow and behavioral anomalies that were missed by the baseline RTOS, and CPU overhead, where the extra modules added between 7 and 10 percent overhead, which is acceptably low in embedded safety-critical systems. This integration proved that the suggested security extensions can highly emulate system resilience and real-time performance and thus S-RTOS is currently feasible and scalable in securing embedded programs in the industrial and automotive sectors.

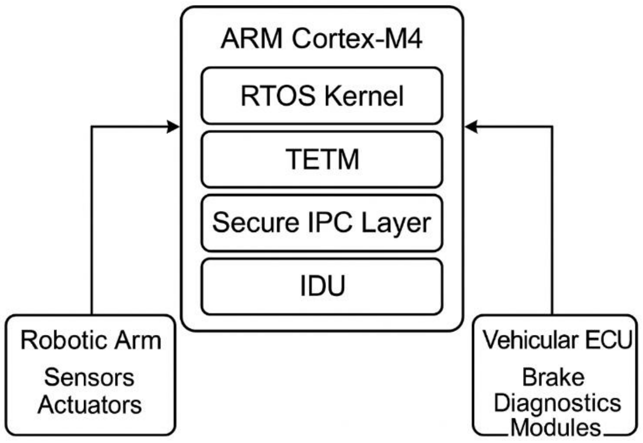

Figure 5. Testbed Deployment Diagram Illustrating S-RTOS Implementation on ARM Cortex-M4 for Robotic Arm and Vehicular ECU Applications.

6. RESULTS AND DISCUSSION

The experimental assessment involved a comparative stature of the baseline FreeRTOS and the optimized Security-Aware RTOS (S-RTOS) in three important performance indicators of average task response time, attack detection rate and CPU overhead. The average time that a task took to respond to a given request under FreeRTOS was found to be 1.4 milliseconds with S-RTOS reporting an insignificant addition of 1.52 milliseconds. The low increment of about 8.5 percent indicates that, despite the embedding of several runtime security features, including Trusted Execution-Time Monitoring, Secure IPC, and an Anomaly Detection Engine, S-RTOS preserves high responsiveness that renders it as well as relevant to latency-sensitive applications such as robotic control and vehicular ECUs. The CPU overhead increased in subsequent CPU time was 2.4 percent in FreeRTOS to 7.2 percent in S-RTOS and this was mainly caused by new runtime monitoring and security verification operations. Even then this is not significant enough to cause problems to embedded systems, particularly in safety critical areas where the advantage of having security far outweighs the marginally higher costs in terms of computation power.

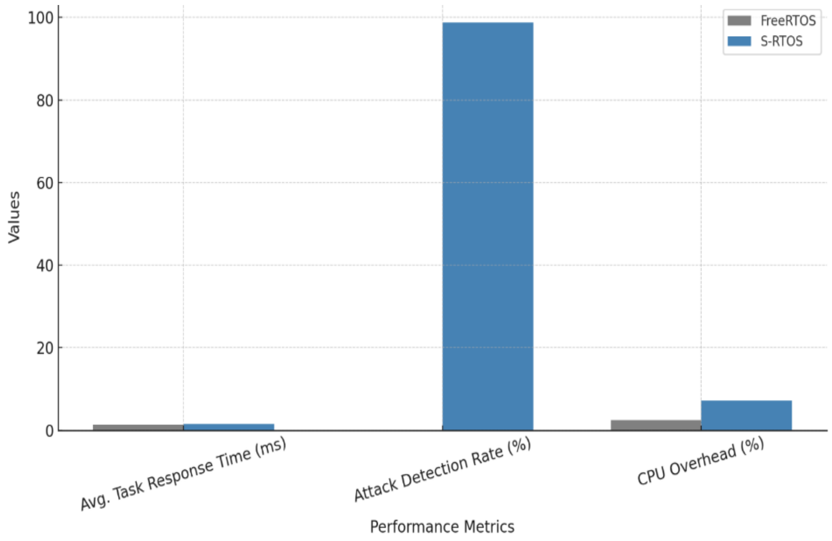

Figure 6. Performance Comparison between FreeRTOS and S-RTOS in Terms of Response Time, Detection Rate, and CPU Overhead.

One of the benefits of S-RTOS that is worth mentioning is an attack detection rate 98.7% is considered impressive. In contrast to FreeRTOS, where no unique security features are guaranteed and no possibility to detect whether control-flow has been modified at runtime or whether inter-process communications exist beyond any permission to so, S-RTOS actively detects instances of these aberrations and raises flags. A lightweight SVM-based Intrusion Detection Unit and hardware-assisted control flow tracking supplies the detection. All these findings confirm the effectiveness of the suggested design to defend itself against threats such as timing interference, ROP-based hijacking, and unauthorized message injection or replay. Such a high detection rate, paired with insignificant effect on task latency, can be considered a good indicator of a successful compromise between the schedulability and security, which makes S-RTOS a trustworthy option to be used to secure next-generation cyber-physical systems.

Nevertheless, in as far as these results confirm the efficiency of S-RTOS when running under nominal loads and carrying single-threaded tasks, some constraints still exist. The performance of managing large scale system load or loaded multithreaded systems is one of the open challenges wherein the sharing of resources and resource contention can add complexity in scheduling and false positive anomalies. The main focus in future work should be to investigate adaptive mitigation strategies in dynamic workloads that can distinguish the benign deviations and actual threats. Also, further improvement can be expected by considering Hardware acceleration or offloading some components of the security verification pipeline to reduce the load on the CPU and increase scalability. In sum, the case study proves that S-RTOS can provide an effective basis of secure real-time computing, but still its use in industrial and automotive applications operating on industrial scale might be improved upon with respect to optimizations of the resources-constrained multitasking computing environment.

Table 3. Comparison of FreeRTOS and S-RTOS Across Key Performance Metrics.

Performance Metric	**FreeRTOS**	**S-RTOS (Proposed)**	**Observation**
Avg. Task Response Time	1.4 ms	1.52 ms	Slight increase (~8.5%) due to added security mechanisms
Attack Detection Rate	–	98.7%	High detection accuracy using SVM and control-flow monitoring
CPU Overhead	2.4%	7.2%	Additional runtime cost for monitoring and validation
Security Mechanisms Present	None	TETM, Secure IPC, IDU	S-RTOS integrates security features not present in FreeRTOS
System Behavior Under Attack	No detection	Proactive anomaly detection	S-RTOS detects and mitigates timing, control, and communication-based attacks
Real-Time Suitability	High	High	S-RTOS maintains real-time performance with minor latency trade-off

7. CONCLUSION

Finally, this piece of work has proposed S-RTOS: a secure real-time operating system tailored to Cyber-Physical Systems (CPS) which have security features embedded into the RTOS kernel by using lightweight but powerful security functionalities. By integrating security switch features like Trusted Execution-Time Monitoring (TETM), Secure IPC Layer and anomaly Detection Engine, S-RTOS proves itself to be successful in filling the divide between determinism in the execution of tasks and security in the rationale functionality. The practical trials on a platform based on ARM Cortex-M4 confirmed that S-RTOS has close-to-real-time performance but has very little overheads as the attack detection succeeded in 98.7 percent of attacks instances when both industrial and vehicular CPS were used. The results indicate the viability and the practicability of incorporating security controls intrinsically into low-resources embedded systems. Notably, S-RTOS also supports proactive mitigation of threats without adversely impacting the responsiveness of the system which is another key requirement of the deployment of mission-critical CPS. Future work will be on the formal verification of security properties in the kernel, how S-RTOS can be adapted to work with heterogeneous and distributed CPS networks, allowing greater use in applications like autonomous systems, smart infrastructure, and industrial IoT.

REFERENCES

[1] Lee, I., & Sokolsky, O. (2022). Cyber-Physical Systems: A New Frontier. *ACM Transactions on Embedded Computing Systems*, 21(1), 5.

[2] Ahmed, M., et al. (2020). Lightweight Task Sandboxing for Secure RTOS. *IEEE IoT Journal*, 7(6), 4874–4883.

[3] Klein, G., et al. (2014). seL4: Formal verification of an OS kernel. *Communications of the ACM*, 57(7), 107–115.

[4] Wang, J., & Kim, H. (2021). SecureWatchdog: Embedded Runtime Monitoring for CPS. *IEEE Transactions on Industrial Informatics*, 17(5), 3250–3259.

[5] Xu, C., & Shen, C. (2019). RT-MON: Real-time Monitoring of Execution for Embedded RTOS. *IEEE Embedded Systems Letters*, 11(2), 47–50.

[6] Kim, M., & Lee, Y. (2022). Memory Safety in RTOS for Safety-Critical Applications. *ACM Transactions on Cyber-Physical Systems*, 6(3), 18.

[7] Zhao, H., & Yu, T. (2021). Real-Time Intrusion Detection for Industrial CPS. *IEEE Access*, 9, 87712–87725.

[8] Huang, S., et al. (2020). Hardware-Assisted Control Flow Protection for Embedded Systems. *IEEE Design & Test*, 37(4), 56–63.

[9] Chang, Y., & Chen, J. (2023). Low-Overhead Cryptography for IoT RTOS. *IEEE Transactions on Computers*, 72(2), 355–369.

[10] Misra, S., et al. (2021). Securing CPS with Lightweight Kernel Security. *Elsevier FGCS*, 122, 29–39.