Multimode Security-Aware Real-Time Scheduling on Multiprocessors

Jiankang Ren, *Member, IEEE*, Chunxiao Liu, Chi Lin, *Senior Member, IEEE*, Wei Jiang, *Senior Member, IEEE*, Pengfei Wang, *Member, IEEE*, Xiangwei Qi, Simeng Li, and Shengyu Li

***Abstract*—Embedded real-time systems generally execute in a predictable and deterministic manner to deliver critical functionality within stringent timing constraints. However, the predictable execution behavior leaves the system vulnerable to schedule-based attacks. In this article, we present a multimode security-aware real-time scheduling scheme to counteract schedule-based attacks on multiprocessor real-time systems. To mitigate the vulnerability to the schedule-based attack, we propose a multimode scheduling method to reduce the accumulative attack effective window (AEW) of multiple victim tasks and prevent the untrusted tasks from executing during the AEW by distinctively scheduling mixed-trust tasks according to the system mode. To avoid the protection degradation due to the excessive blocking of untrusted tasks, we introduce a protection window for multiple victims on multiprocessors by analyzing the system protection capability limit under the system schedulability constraint. Furthermore, to maximize the protection capability of the multimode security-aware scheduling strategy on a multiprocessor platform, we also propose a security-aware packing algorithm to balance the workloads of mixed-trust tasks on different processors using a mixed-trust worst-fit decreasing heuristic strategy. The experimental results demonstrate that our proposed approach significantly outperforms the state-of-the-art method. Specifically, the AEW ratio and the AEW untrusted execution time ratio are reduced by 18.8% and 62.8%, respectively, while the defense success rate against ScheduLeak attack is improved by 16.3%.**

***Index Terms*—Multimode scheduling, multiprocessor, real-time systems, schedule-based attacks, security-aware scheduling.**

Manuscript received 5 August 2024; accepted 10 August 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62072067, Grant 62172069, Grant 62072076, Grant 62466059, and Grant 61602080; in part by the Shandong Provincial Natural Science Foundation under Grant ZR2023LZH016; and in part by the Xinjiang Network Information Science and Technology Innovation Research Project under Grant 12421604. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Chi Lin.)*

Jiankang Ren, Chunxiao Liu, Pengfei Wang, Simeng Li, and Shengyu Li are with the Key Laboratory of Social Computing and Cognitive Intelligence, Ministry of Education, Dalian University of Technology, Dalian 116024, China (e-mail: rjk@dlut.edu.cn; liuchuanxiao@mail.dlut.edu.cn; wangpf@dlut.edu.cn; lsmhf@mail.dlut.edu.cn; lsyanling@mail.dlut.edu.cn).

Chi Lin is with the School of Software Technology, Dalian University of Technology, Dalian 116024, China (e-mail: c.lin@dlut.edu.cn).

Wei Jiang is with the School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China (e-mail: weijiang@uestc.edu.cn).

Xiangwei Qi is with the School of Computer Science and Technology, Xinjiang Normal University, Xinjiang 830054, China (e-mail: xiangweiqi10@163.com).

Digital Object Identifier 10.1109/TCAD.2024.3445260

I. Introduction

EMBEDDED real-time systems are extensively applied in safety-critical applications, such as automotive, aviation, and industrial robotics. These systems generally execute in a predictable and deterministic manner to deliver critical functionality within stringent timing constraints. However, such a predictable execution pattern exposes a vulnerability that can be exploited by schedule-based attacks [1]. For example, adversaries can leverage the deterministic execution patterns to infer sensitive scheduling information through timing side-channels [2]. With knowledge of system internals gleaned from such attacks, malicious attackers can craft more effective tailored attacks, such as compromising system stability through the injection of inaccurate data [3] and affecting vehicle operations by obstructing control system signals [2], [4]. Moreover, the relentless demand for computational power has driven embedded real-time systems toward multiprocessor architectures. Consequently, it is extremely important to offer an effective security strategy to defend against schedule-based attacks for multiprocessor real-time systems without compromising real-time performance constraints.

The effectiveness of schedule-based attacks typically depends on whether the attacker task is executed during the attack effective windows (AEWs) of victim tasks [5]. For example, the experimental evidence has shown that the AEW for a control output overwrite attack on a customized rover system with real-time Linux is 8.3 ms [2]. According to the timing relationships between the execution states of the victim and the attacker, schedule-based attacks can be divided into four categories [6]: 1) the posterior attack that is launched after the completion of the victim task; 2) the anterior attack that is carried out before the victim task is executed; 3) the concurrent attack that takes place while the victim task is being executed; and 4) the pincer attack that is a hybrid attack combining both posterior and anterior attacks. In this article, we focus on mitigating the posterior schedule-based attack. Note that, as a common threat to real-time systems, the posterior schedule-based attacks, such as replay attacks, zero dynamics attacks, and bias injection attacks, can be launched to manipulate the output of a task, and their attack effects have been demonstrated in Real-time Linux and FreeRTOS [2], [3].

There is a wealth of literature on security-aware scheduling methods to defend against schedule-based attacks (outlined in Section II). Traditionally, research in this field has focused mainly on uniprocessor systems [5], [7], [8], [9], [10], but lately, attention has turned toward multiprocessor systems [11], [12]. More recently, Chen et al. [11] proposed a temporal isolation-based protection approach *SchedGuard++* to protect against posterior schedule-based attacks on multiprocessors with multiple victims by preventing untrusted tasks from running within AEWs of victim tasks. It can provide best-effort protection for the multiprocessor system under schedulability constraints. However, it has four major drawbacks.

1) It tends to allocate all untrusted tasks to the same processors, thereby reducing opportunities to mitigate schedule-based attacks by strategically scheduling untrusted tasks with parallel processing capabilities of multiprocessors.
2) Upon the completion of a victim task's execution on a processor, it blocks all other processors, including those running trusted tasks. This resource wastage leads to a decline in the performance of security protections under the system schedulability constraint.
3) It employs an empirical protection window to guide the security-aware scheduling, neglecting the system protection capability limit imposed by schedulability constraints. This may result in over blocking of untrusted tasks, ultimately compromising the performance of the security protection.
4) It conducts an offline analysis for the maximum tolerable blocking times of tasks, ignoring the run-time system behavior. The inherent pessimism of offline analysis results in a degradation of protection performance, particularly when handling task sets with high utilization.

To overcome the above limitations, we propose a multimode security-aware real-time scheduling approach called MM-SARTS for multiprocessor real-time systems to optimize the system protection capability under system schedulability constraints. MM-SARTS enables the system to operate in different modes, each with its own specific security-aware scheduling strategy, to mitigate schedule-based attacks by distinctively scheduling mixed-trust tasks according to the system's operational status. The primary contributions of our work can be summarized as follows.

1) We presented an example to illustrate the limitations of the isolation-based protection method *SchedGuard++* for multiprocessor systems with multiple victims.
2) We proposed a multimode security-aware real-time scheduling method to mitigate schedule-based attacks by distinctively scheduling mixed-trust tasks according to the system mode, coupled with an online priority inversion feasibility test.
3) We introduced a protection window for multiple victims on multiprocessors to avoid protection degradation due to excessive blocking of untrusted tasks by analyzing the system protection capability limit under the system schedulability constraint.
4) We developed a security-aware task-to-processor packing algorithm that maximizes the protection capability of the multimode security-aware scheduling strategy on a multiprocessor system by balancing the workloads of mixed-trust tasks across different processors.

The experimental evaluation based on an automotive benchmark indicates that our method can effectively decrease the AEW ratio and the AEW untrusted execution time ratio and enhance the attack defense success rate.

II. Related Work

Since the pioneer research by Son et al. [13] on information leakage in real-time systems caused by predictable system execution patterns, numerous studies have focused on security-aware real-time scheduling strategies to counter schedule-based attacks. These security-aware scheduling techniques can generally be categorized into two groups: 1) randomization-based scheduling and 2) isolation-based scheduling.

In randomization-based scheduling, obfuscation mechanisms are used to diversify the schedule, making it difficult for attackers to accurately predict the timing behavior of victim tasks [7], [8], [9], [14], [15], [16], [17]. For the fixed-priority real-time system in which each task is assigned a static priority level, Yoon et al. [7] introduced a randomized security-aware scheduling method based on priority inversion. This method leverages statically calculated priority inversion budgets to resist the schedule-based side-channel attacks while ensuring the real-time performance. To increase randomness in the task execution pattern, Yoon et al. [14] utilized runtime information at each scheduling decision point to enhance the uncertainty of task schedules while ensuring system schedulability. For the time-triggered real-time system, where tasks are executed according to a predetermined and fixed schedule constructed based on the schedulability constraints, Krüger et al. [15], [16] analyzed vulnerabilities related to timing inference and malicious behavior and proposed an online job randomization method and an offline schedule-diversification method to mitigate timing inference-based attacks. For the dynamic-priority real-time system, where task priorities are calculated during system execution, Chen et al. [8] introduced a randomized scheduling method to obscure the earliest deadline first scheduling policy with limited priority inversions at run-time. This method effectively introduces unpredictability into execution patterns of tasks, particularly when the system operates under low and medium loads. However, its conservative predetermination of task priority inversion limits, without considering dynamic run-time system behavior, can lead to performance degradation under heavy utilization. To address this problem, Ren et al. [9] proposed an enhanced randomized scheduling strategy that leverages runtime system information to increase feasible priority inversion opportunities while ensuring system schedulability. Although randomization-based scheduling methods can significantly increase timing uncertainty, making it harder to predict task execution states, it has been demonstrated that they may fail to protect against certain schedule-based attacks and, in some cases, even increase the attack success rate [6].

In isolation-based scheduling, various temporal isolation mechanisms are employed to prevent interference among tasks, thereby protecting sensitive information from unauthorized access [5], [10], [18], [19], [20]. For the fixed-priority real-time systems, Völp et al. [18] suggested using an idle system thread to defend against information leakage. Similarly, Pellizzoni et al. [20] and Mohan et al. [19] suggested the introduction of flush tasks to prevent the schedule-based information leakage between low- and high-security tasks. However, this mechanism introduces significant overhead, yielding in a poor response time for real-time tasks and reducing system schedulability. To prevent the execution of untrusted tasks during the AEWs, Chen et al. [5] proposed a coverage-oriented scheduling policy to provide deterministic isolation against posterior schedule-based attacks without affecting schedulability. However, this approach overlooks the limit of system protection capacity due to schedulability constraints in security-aware scheduling decisions, potentially leading to poor security performance from excessive blocking of untrusted tasks. Moreover, it conducts an offline analysis of the maximum acceptable blocking time budget, which results in diminished protection performance when handling task sets with high utilization due to the pessimism of the offline analysis. To avoid the excessive blocking of untrusted tasks and the pessimism of the offline analysis, Ren et al. [10] proposed a security-aware real-time scheduling scheme based on the protection window and the online feasibility test. However, this method is limited to single-core systems with a single victim task. For multiprocessor platforms with multiple victims, Chen et al. [11] introduced an approach named *SchedGuard++*, which extends the coverage-oriented scheduling approach in [5] with the worst-case response time analysis for mixed-trust tasks on the multiprocessor. Nevertheless, it fails to fully exploit the parallelism of multiprocessor systems to optimize security performance under schedulability constraints by decreasing the AEW ratio and the AEW untrusted execution time ratio.

In this article, we propose a novel isolation-based security-aware scheduling approach for multiprocessor systems with multiple victims. Our approach differs from these existing methods in that it can effectively reduce the AEW ratio and the AEW untrusted execution time ratio through multimode scheduling based on an online priority inversion feasibility test. Furthermore, our approach enhances the overall system protection capability of multiprocessor systems by balancing mixed-trust task workloads across different processors.

III. System and Adversary Model

A. System Model

We consider a real-time system comprising N independent periodic real-time tasks, denoted by $\Gamma = \{\tau_1, \ldots, \tau_N\}$, executed on a multiprocessor system with P identical processors of unit capacity, identified as $\Pi = \{\pi_1, \ldots \pi_P\}$, following a partitioned fixed-priority preemptive scheduling strategy commonly used in OSEK/VDX operating system [21] and AUTOSAR [22]. Each task $\tau_i \in \Gamma$ is defined as $\tau_i = (C_i, T_i, D_i)$, where

1) C_i is the worst-case execution time (WCET) of τ_i;
2) T_i is the period of τ_i;
3) D_i is the relative deadline of τ_i.

For a task τ_i, we use *utilization* symbolized by U_i to indicate the ratio C_i/T_i, and use $U_\Gamma = \sum_{\tau_i \in \Gamma} U_i$ to represent the *total utilization* of the task set Γ. We consider all tasks to be implicit-deadline tasks, where the deadline D_i is equal to the period T_i, and they are initially released at the same time $t = 0$. The hyperperiod of a task set Γ is indicated by H_Γ, which is the least common multiple of all task periods for task set Γ. The job of task τ_i is represented as $\mathcal{J}_{i,j}$, and its release time is indicated as $r_{i,j}$, which is a member of the infinite set $\{0, T_i, 2T_i, \ldots\}$. For a job $\mathcal{J}_{i,j}$ of task τ_i released at $r_{i,j}$, its completion time is represented as $f_i(r_{i,j})$. If each task $\tau_i \in \Gamma$ can meet its deadline in the worst-case scenario, the system is considered schedulable. For each task $\tau_i \in \Gamma$, it has a unique priority, and its priority is allocated based on the rate monotonic (RM) policy [23]. The set of tasks with priorities higher than task τ_i is denoted as $hp(\tau_i)$, while the set of tasks with lower priorities is indicated as $lp(\tau_i)$. Following [5], the idle times are treated as instances of an extra idle task in the system, and the idle task has the lowest priority, infinite execution time, and infinite period and deadline.

B. Adversary Model

In this article, we follow the vendor-oriented security model in [20], where information leakage from a vendor's sensitive tasks to other vendors' tasks is undesirable. For a system, high-critical tasks (e.g., engine and brake control tasks in a self-driving system) are regarded as victim tasks. Given a victim task set, tasks are classified as trusted (from the same vendor as a victim task, or an idle task) or untrusted (all other tasks, which may be attackers). It is assumed that only untrusted tasks pose an attack risk, and the scheduler is trustworthy.

We consider an attack scenario where an adversary carries out a posterior schedule-based attack on the victim task by exploiting external connections on the target platform [6]. It is assumed that the adversary has taken control of certain tasks, turning them into attackers within the system, and is able to modify their control flow. The attacker is unaware of the concrete scheduling scheme, but it can deduce certain scheduling parameters by monitoring the execution windows of compromised tasks. For example, such an attack can covertly deduce the locations and routes of self-driving cars through the external network [4]. We assume that for the attack to be effective, it must be carried out within a certain time window after the victim task completes to steal, corrupt, or overwrite the victim's output. We define such a time window as the AEW.

Definition 1 [11]: For a victim task τ_i^{V}, its AEW, denoted by ω_i, is defined as the time period during which schedule-based attacks are effective and ineffective otherwise.

To characterize the amount of AEWs generated by a victim task within a time interval, we define the accumulative AEWs of a victim task as follows.

Definition 2 [11]: Given a scheduling policy $\mathcal{P}$ and a schedulable task set Γ under $\mathcal{P}$, for a victim task $\tau_i^{\mathsf{V}} \in \Gamma$, its *accumulative AEW* within the time interval $[t_1, t_2)$, denoted by $\Omega(\{\tau_i^{\mathsf{v}}\}, \mathcal{P}, t_1, t_2)$, is defined as the set of all time intervals that belong to the AEW of task τ_i^{v} within time interval $[t_1, t_2)$.

We focus on multiprocessor real-time systems with multiple victim tasks, and the AEWs of different victim tasks can potentially overlap because of the parallel execution on different processors. Here, we formally define the accumulative AEW for a set of victim tasks as the union of their AEWs.

Definition 3 [11]: Given a scheduling policy $\mathcal{P}$ and a schedulable task set Γ under $\mathcal{P}$, for the victim task set $\Gamma^{\mathsf{v}} \subseteq \Gamma$, its *accumulative AEW* within the time interval $[t_1, t_2)$, denoted by $\Omega(\Gamma^{\mathsf{v}}, \mathcal{P}, t_1, t_2)$, is defined as the union of AEWs of all victim tasks in Γ^{v} over the time interval $[t_1, t_2)$, i.e.,

$$\Omega\big(\Gamma^{\mathsf{v}}, \mathcal{P}, t_1, t_2\big) = \bigcup_{\tau_i^{\mathsf{v}} \in \Gamma^{\mathsf{v}}} \Omega\big(\{\tau_i^{\mathsf{v}}\}, \mathcal{P}, t_1, t_2\big) \tag{1}$$

where $\Omega(\{\tau_i^{\mathsf{v}}\}, \mathcal{P}, t_1, t_2)$ is the accumulative AEW of task τ_i^{v} over the time interval $[t_1, t_2)$ under scheduling policy $\mathcal{P}$.

To assess the protection performance of a scheduling policy, we define the AEW ratio and the AEW untrusted execution time ratio as follows.

Definition 4: Given a scheduling policy $\mathcal{P}$ and a schedulable task set Γ under $\mathcal{P}$, *the AEW ratio* of the task set Γ under $\mathcal{P}$ within the time interval $[t_1, t_2)$, denoted by $\Lambda(\Gamma, \mathcal{P}, t_1, t_2)$, is defined as

$$\Lambda(\Gamma, \mathcal{P}, t_1, t_2) = \frac{L\big(\Gamma^{\mathsf{v}}, \mathcal{P}, t_1, t_2\big)}{t_2 - t_1} \tag{2}$$

where Γ^{v} represents all victim tasks in Γ, and $L(\Gamma^{\mathsf{v}}, \mathcal{P}, t_1, t_2)$ is the cumulative length of the accumulative AEW $\Omega(\Gamma^{\mathsf{v}}, \mathcal{P}, t_1, t_2)$ of task set Γ^{v} within the time interval $[t_1, t_2)$.

From Definition 4, we can observe that for a task set Γ, a higher AEW ratio indicates a larger attack surface, thereby increasing its susceptibility to attacks.

Definition 5: Given a scheduling policy $\mathcal{P}$ and a schedulable task set Γ under $\mathcal{P}$, *the AEW untrusted execution time ratio* for the task set Γ within the time interval $[t_1, t_2)$, denoted by $\Theta(\Gamma, \mathcal{P}, t_1, t_2)$, is defined as the total execution time of untrusted tasks within AEWs in $[t_1, t_2)$ divided by their cumulative execution time within the time interval $[t_1, t_2)$, i.e.,

$$\Theta(\Gamma, \mathcal{P}, t_1, t_2) = \frac{\sum_{\tau_i \in \Gamma^{\mathsf{u}}} E_i^{\mathsf{AEW}}(\mathcal{P}, t_1, t_2)}{\sum_{\tau_i \in \Gamma^{\mathsf{u}}} E_i(\mathcal{P}, t_1, t_2)} \tag{3}$$

where Γ^{u} is the set of all untrusted tasks in Γ, $E_i^{\mathsf{AEW}}(\mathcal{P}, t_1, t_2)$ represents the total execution time of the untrusted task $\tau_i \in \Gamma^{\mathsf{u}}$ within AEWs in $[t_1, t_2)$ under scheduling policy $\mathcal{P}$ and $E_i(\mathcal{P}, t_1, t_2)$ denotes the cumulative execution time of task τ_i within $[t_1, t_2)$ under scheduling policy $\mathcal{P}$.

From Definition 5, we can see that for a task set Γ, as the AEW untrusted execution time ratio increases, the execution time of untrusted tasks within AEWs tends to be longer, thus resulting in a higher likelihood of successful attacks.

Goal: The main objective of this work is to develop an efficient security-aware multiprocessor real-time scheduling strategy to schedule a given set of periodic real-time tasks on the multiprocessor platform with multiple victim tasks, such that all tasks are schedulable and the chance for the adversary to launch a successful posterior scheduler-based attack is reduced by minimizing the AEW ratio and the AEW untrusted execution time ratio.

TABLE I
TASK PARAMETERS OF AN EXAMPLE TASK SET Γ

Task	C_i	T_i	D_i	Trust Level	ω_i
τ_1^{v}	1	10	10	Trusted	3
τ_2^{v}	2	20	20	Trusted	5
τ_1^{t}	1	10	10	Trusted	-
τ_2^{t}	2	10	10	Trusted	-
τ_3^{t}	4	20	20	Trusted	-
τ_4^{t}	4	20	20	Trusted	-
τ_1^{u}	4	10	10	Untrusted	-
τ_2^{u}	4	10	10	Untrusted	-
τ_3^{u}	5	20	20	Untrusted	-
τ_4^{u}	5	20	20	Untrusted	-

IV. MULTIMODE SECURITY-AWARE REAL-TIME SCHEDULING

A. Motivation

Before presenting our multimode security-aware real-time scheduling scheme, we first provide an example to discuss the limitations of the existing temporal isolation-based protection method *SchedGuard*++ [11] and to motivate the multimode security-aware real-time scheduling strategy. The basic idea of *SchedGuard*++ can be succinctly described as follows.

1) In the task-to-processor allocation process, each victim task is initially allocated to a single processor. Next, trusted tasks are evenly distributed among processors with victim tasks based on their utilizations. Finally, untrusted tasks are distributed evenly among processors without victim tasks. It is important to note that, to assign a feasible processor to each task, trusted and untrusted tasks may be allocated to any processor, regardless of whether that processor has a victim task.
2) It schedules tasks on the basis of an empirical protection window. Once a victim task is completed on a processor, the protection window starts. During the protection window, all other processors are attempted to be blocked. Note that when a task reaches its maximum tolerable blocking time (i.e., the longest duration that a task can be paused or delayed by lower priority tasks under the schedulability constraint), it is allowed to execute within the protection window to ensure system schedulability.

It has been demonstrated that *SchedGuard*++ can defend against posterior schedule-based attacks on multiprocessors by preventing untrusted tasks from being executed during the AEW. However, *SchedGuard*++ cannot effectively reduce the AEW ratio and the AEW untrusted execution time ratio for some sets of tasks. Now, we illustrate this with an example.

Example 1: Consider a task set Γ depicted in Table I scheduled on four processors $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ under *SchedGuard*++. As shown in Fig. 1, it is the simulation of a synchronous arrival sequence (SAS) for the task set Γ under *SchedGuard*++. From Fig. 1, we can see that all untrusted tasks are allocated to processors π_3 and π_4, although there remains some capacity on processors π_1 and π_2. This reduces opportunities to mitigate the schedule-based attack

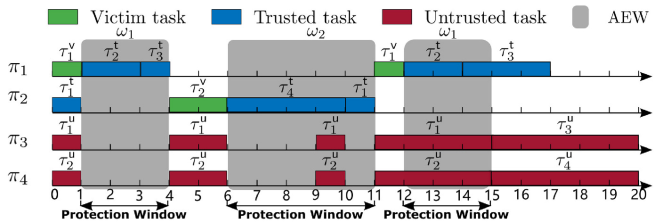

Fig. 1. SAS simulation for task set Γ under *SchedGuard++*.

by strategically scheduling untrusted tasks with the parallel processing characteristics of multiprocessors. We also can observe that once a job of victim task τ_1^{v} is completed, all other processors, including the processor π_2 with trusted tasks, are attempted to be blocked. This results in a reduction in the system's parallel processing capability and an increase in the AEW ratio. Obviously, some untrusted jobs are executed during AEWs of victim tasks τ_1^{v} and τ_2^{v}. From Fig. 1, we can see that the accumulative AEW within $[0, 20)$ is $3 + 5 + 3 = 11$, and hence the AEW ratio within $[0, 20)$ is $11/20 = 0.55$. We also can see that the total execution time of untrusted tasks within AEWs in $[0, 20)$ is $2+6 = 8$ and the cumulative execution time of untrusted tasks within $[0, 20)$ is $2 \times 4+2 \times 4 + 5 + 5 = 26$, and thus the AEW untrusted execution time ratio within $[0, 20)$ is $8/26 \approx 0.3077$. Note that it is possible to decrease the AEW ratio within $[0, 20)$ to 0.45 and the AEW untrusted execution time ratio within $[0, 20)$ to 0 using our multimode security-aware real-time scheduling approach (see Example 2).

B. Multimode System Model

To mitigate the vulnerability to schedule-based attacks, we model the real-time system as a multimode system and introduce an enforcement mechanism to prevent untrusted tasks from being executed during AEWs of victim tasks by scheduling specific jobs to execute based on the system mode.

Multimode System: The real-time system is modeled as a multimode system characterized by a set of system modes $\mathcal{M}$, an initial system mode $\mathsf{M}_0 \in \mathcal{M}$, a set of mode transitions $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$, and a set of implicit-deadline periodic tasks $\mathcal{T}$ that execute in the system modes. Here, we consider three system modes: 1) normal mode M^{N}; 2) victim mode M^{V}; and 3) protection mode M^{P}, with the initial mode being the normal mode (i.e., $\mathsf{M}_0 = \mathsf{M}^{\mathsf{N}}$). For each mode $\mathsf{M} \in \mathcal{M}$, we consider that all tasks in the set $\mathcal{T}$ should be executed. The mode transition is triggered by a mode change request event (MCR).

Enforcement in a Mode: To mitigate schedule-based attacks by strategically scheduling mixed-trust tasks based on the system's operational mode, we define the scheduling enforcement for each system mode as follows.

1) *Normal Mode* M^{N}*:* Untrusted tasks are executed as much as possible on each processor. This enables the system to execute untrusted tasks to the greatest extent possible before the execution of victim tasks, preventing untrusted tasks from launching posterior schedule-based attacks.
2) *Victim Mode* M^{V}*:* Victim tasks are executed as much as possible on each processor. This allows victim tasks on different processors to execute simultaneously, maximizing the overlap of AEWs of victim tasks across processors and thereby reducing the accumulative AEW size of multiple victim tasks in the multiprocessor system.
3) *Protection Mode* M^{P}*:* Trusted and idle tasks are executed as much as possible on each processor. With this mode, we can minimize the risk of potential attacks on victim tasks by preventing untrusted tasks from being executed during AEWs of victim tasks.

Note that the mode enforcement is *nonstrict*. When a task must be executed for schedulability, regardless of its type, it can be executed in any mode to maintain the system schedulability.

Enforcement Upon an MCR: When the system is operating in mode M_s and receives an MCR associated with an outgoing transition $(\mathsf{M}_s, \mathsf{M}_t)$, it immediately switches to the new mode M_t and performs the enforcement. We consider four mode transitions.

1) $\mathsf{M}^{\mathsf{N}} \rightarrow \mathsf{M}^{\mathsf{V}}$: This transition allows the system to execute untrusted tasks prior to the execution of victim tasks. It is triggered when no untrusted tasks are pending on any processor, or when there is a victim task that must be executed for schedulability.
2) $\mathsf{M}^{\mathsf{V}} \rightarrow \mathsf{M}^{\mathsf{P}}$: This transition enables the system to execute trusted tasks after the completion of victim tasks. It is triggered when no victim tasks are pending on any processor.
3) $\mathsf{M}^{\mathsf{P}} \rightarrow \mathsf{M}^{\mathsf{N}}$: This transition is used to avoid the protection degradation due to the excessive blocking of untrusted tasks. It is triggered when the duration of the protection mode exceeds a specific threshold. This threshold is set based on the system protection capability limit characterized by the protection window given in Section IV-C.
4) $\mathsf{M}^{\mathsf{P}} \rightarrow \mathsf{M}^{\mathsf{V}}$: This transition is designed to ensure the schedulability of the victim tasks and provide timely protection for them. It is triggered when a victim task must be executed for schedulability.

C. Protection Window of Multiple Victims on Multiprocessor

To effectively mitigate schedule-based attacks by preventing the protection degradation caused by the excessive blocking of untrusted tasks in the protection mode, we introduce the protection window for multiple victim tasks on the multiprocessor platform. This protection window characterizes the limit of system protection capability under schedulability constraints.

Definition 6: Given a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k, *the protection window of a victim job* $\mathcal{J}_{i,j}^{\mathsf{v}}$ *of victim task* $\boldsymbol{\tau_i^{\mathsf{v}} \in \Gamma(\pi_k)}$, denoted by $S_{i,j}$, is defined as the length of time interval $[f^{\mathsf{v}}(r_{i,j}^{\mathsf{v}}), \min\{r_{i,j}^{\mathsf{v}} + T_i^{\mathsf{v}}, b^{\mathsf{u}}\})$, where $f^{\mathsf{v}}(r_{i,j}^{\mathsf{v}})$ is the finish time of the job $\mathcal{J}_{i,j}^{\mathsf{v}}$, T_i^{v} is the period of task τ_i^{v}, and b^{u} is the start time of execution of the first untrusted job executed after $f^{\mathsf{v}}(r_{i,j}^{\mathsf{v}})$.

Definition 7: Given a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k, *the protection window of a victim task* $\boldsymbol{\tau_i^{\mathsf{V}}} \in \boldsymbol{\Gamma(\pi_k)}$, denoted by $\mathcal{S}_i$, is defined as the minimum protection window of jobs for victim task τ_i^{V}, i.e.,

$$\mathcal{S}_i = \min_{1\le j\le H_{\Gamma(\pi_k)}/T_i^{\mathsf{V}}} \{S_{i,j}\} \tag{4}$$

where $S_{i,j}$ is the protection window of job $\mathcal{J}_{i,j}^{\mathsf{V}}$ for τ_i^{V}, $H_{\Gamma(\pi_k)}$ is the hyperperiod of $\Gamma(\pi_k)$, and T_i^{V} is the period of τ_i^{V}.

Lemma 1: Given a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k, the protection window of a victim task $\tau_i^{\mathsf{V}} \in \Gamma(\pi_k)$ is bounded by

$$\mathcal{S}_i^{\max} = T_i^{\mathsf{V}}\left(1 - \left(\sum_{\tau_j\in\Gamma^{\mathsf{U}}(\pi_k)} U_j\right) - U_i^{\mathsf{V}}\right) \tag{5}$$

where $\Gamma^{\mathsf{U}}(\pi_k)$ is the set of all untrusted tasks in task set $\Gamma(\pi_k)$, T_i^{V} is the period of task τ_i^{V}, and U_j and U_i^{V} represent the utilizations of tasks τ_j and τ_i^{V}, respectively.

Proof: For a hyperperiod with $m = H_{\Gamma(\pi_k)}/T_i^{\mathsf{V}}$ jobs of the victim task τ_i^{V}, where $H_{\Gamma(\pi_k)}$ is the hyperperiod of task set $\Gamma(\pi_k)$ and T_i^{V} is the period of task τ_i^{V}, we can derive the following inequality based on Definition 7:

$$m\mathcal{S}_i \le \sum_{1\le l\le m} S_{i,l} \tag{6}$$

where $S_{i,l}$ is the protection window of victim job $\mathcal{J}_{i,l}^{\mathsf{V}}$, and $\mathcal{S}_i$ is the protection window of victim task τ_i^{V}.

According to Definition 6, there is no execution of any untrusted task or victim task τ_i^{V} within the protection window of the job of victim task τ_i^{V}, and hence we can derive

$$\begin{aligned}\sum_{1\le l\le m} S_{i,l} &\le H_{\Gamma(\pi_k)} - \left(\sum_{\tau_j\in\Gamma^{\mathsf{U}}(\pi_k)} \frac{H_{\Gamma(\pi_k)}}{T_j} C_j\right) - \frac{H_{\Gamma(\pi_k)}}{T_i^{\mathsf{V}}} C_i^{\mathsf{V}} \\ &= mT_i^{\mathsf{V}}\left(1 - \left(\sum_{\tau_j\in\Gamma^{\mathsf{U}}(\pi_k)} U_j\right) - U_i^{\mathsf{V}}\right).\end{aligned} \tag{7}$$

From (6) and (7), we can derive the following:

$$\mathcal{S}_i \le T_i^{\mathsf{V}}\left(1 - \left(\sum_{\tau_j\in\Gamma^{\mathsf{U}}(\pi_k)} U_j\right) - U_i^{\mathsf{V}}\right). \tag{8}$$

Thus, the protection window of the victim task $\tau_i^{\mathsf{V}} \in \Gamma(\pi_k)$ is bounded by $\mathcal{S}_i^{\max}$ given in (5). ∎

Definition 8: Given a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k, *the protection window of all victim tasks in task set* $\boldsymbol{\Gamma(\pi_k)}$, denoted by $\mathcal{S}(\pi_k)$, is defined as the minimum protection window of victim tasks in task set $\Gamma(\pi_k)$, i.e.,

$$\mathcal{S}(\pi_k) = \min_{\tau_i^{\mathsf{V}}\in\Gamma^{\mathsf{V}}(\pi_k)} \{\mathcal{S}_i\} \tag{9}$$

where $\Gamma^{\mathsf{V}}(\pi_k)$ is the set of all victim tasks in task set $\Gamma(\pi_k)$, and $\mathcal{S}_i$ is the protection window of victim task $\tau_i^{\mathsf{V}} \in \Gamma^{\mathsf{V}}(\pi_k)$.

Note that for a processor without a victim task, we assume its protection window to be infinite. Based on Definition 8 and Lemma 1, we can directly derive the following lemma.

Algorithm 1 Multimode Security-Aware Scheduling

Input: Scheduling point t, current system mode M_t, task set $\Gamma(\pi_k)$, ready queue Q_t, victim ready queue Q_t^{V}, trusted ready queue Q_t^{T}, untrusted ready queue Q_t^{U}.
Output: A job executed at time t.

```
if M_t = M^N then
    J_test ← the highest priority job in Q_t^U
else if M_t = M^V then
    J_test ← the highest priority job in Q_t^V
else
    if Q_t^T ≠ ∅ then
        J_test ← the highest priority job in Q_t^T
    else
        J_test ← idle job
    end if
end if
if J_test is the highest priority job in Q_t then
    J_test is executed until the next scheduling point.
else
    (flag, E_test^t) ← FeasibilityTest(Γ(π_k), J_test, t)
    if flag = True then
        J_test is executed until the next scheduling point.
    else
        The highest priority job in Q_t is executed until the next
        scheduling point.
    end if
end if
```

Lemma 2: Given a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k, the protection window $\mathcal{S}(\pi_k)$ of all victim tasks in task set $\Gamma(\pi_k)$ is bounded by

$$\mathcal{S}^{\max}(\pi_k) = \min_{\tau_i^{\mathsf{V}}\in\Gamma^{\mathsf{V}}(\pi_k)} \left\{\mathcal{S}_i^{\max}\right\} \tag{10}$$

where $\Gamma^{\mathsf{V}}(\pi_k)$ is the set of all victim tasks in task set $\Gamma(\pi_k)$, and $\mathcal{S}_i^{\max}$ is the protection window upper bound of task $\tau_i^{\mathsf{V}} \in \Gamma^{\mathsf{V}}(\pi_k)$ given in (5).

Definition 9: Given a schedulable task set Γ scheduled on a multiprocessor Π with a partitioned scheduling policy, the *protection window of all victim tasks in the task set* $\boldsymbol{\Gamma}$, denoted by $\mathcal{S}_{\Pi}$, is defined as follows:

$$\mathcal{S}_{\Pi} = \min_{\pi_k\in\Pi} \{\mathcal{S}(\pi_k)\} \tag{11}$$

where $\mathcal{S}(\pi_k)$ is the protection window of all victim tasks in the task set $\Gamma(\pi_k)$ assigned to processor $\pi_k \in \Pi$.

Based on Definition 9 and Lemma 2, the following theorem can be directly derived.

Theorem 1: Given a schedulable task set Γ scheduled on a multiprocessor Π with a partitioned scheduling policy, the protection window of all victim tasks in Γ is bounded by

$$\mathcal{S}_{\Pi}^{\max} = \min_{\pi_k\in\Pi} \left\{\mathcal{S}^{\max}(\pi_k)\right\} \tag{12}$$

where $\mathcal{S}^{\max}(\pi_k)$ is the protection window upper bound of tasks assigned to the processor $\pi_k \in \Pi$ given in (10).

For a task set Γ scheduled on a multiprocessor Π with a partitioned scheduling policy, once the duration of the protection mode M^{P} exceeds $\mathcal{S}_{\Pi}^{\max}$, the system will immediately enter the normal mode M^{N} to avoid the protection degradation due to excessive blocking of untrusted tasks.

D. Multimode Security-Aware Scheduling

Based on the multimode system model given in Section IV-B we propose a multimode security-aware real-time scheduling algorithm to provide best-effort security protection for victim tasks while maintaining system schedulability. The key idea of this scheduling algorithm is to reduce the accumulative AEW of multiple victim tasks and prevent untrusted tasks from executing during the AEW by distinctively scheduling mixed-trust tasks according to the system mode.

As given in Algorithm 1, it is our multimode security-aware scheduling algorithm written in pseudocode. In the algorithm, there is a ready queue Q_t that holds all jobs that are ready to run and waiting to be executed, with Q_t^{V}, Q_t^{T}, and Q_t^{U} representing the ready queues for victim jobs, trusted jobs, and untrusted jobs, respectively. The scheduling decisions are made by selecting a candidate job according to the system mode. When the system is in normal mode, the execution feasibility of the ready untrusted job with the highest priority will be checked (lines 1 and 2). When the system is in victim mode, the execution feasibility of the ready victim job with the highest priority will be checked (lines 3 and 4). When the system mode is in protection mode, if there are some trusted jobs in the ready queue, the execution feasibility of the ready trusted job with the highest priority will be checked (lines 6 and 7); otherwise, the execution feasibility of the idle job will be checked (line 9). Note that it is always feasible for the highest priority job in the ready queue Q_t, since there is no priority inversion for its execution (lines 12 and 13). If $\mathcal{J}_{\text{test}}$ is not the highest priority job in ready queue Q_t, an online priority inversion feasibility test is performed for $\mathcal{J}_{\text{test}}$, and the online feasibility test algorithm is given in Algorithm 2. If it is feasible to execute the selected job, the selected job will be executed at t (lines 16 and 17); otherwise, the highest priority job in the ready queue Q_t will be executed (line 19). Note that the selected job is executed until the next scheduling point. In our scheduling algorithm, the scheduling points include the arrival time of the MCR associated with a mode translation, the completion time of the current job, the moment when the running job experienced the feasible execution time E_{test}^t, and the release instant of new jobs.

From Algorithm 1, we can find that some higher priority tasks can be blocked by lower priority tasks (i.e., priority inversion) during the security-aware scheduling of each system mode. To ensure system schedulability in the presence of priority inversion, we conduct an online feasibility test based on the priority inversion budget analysis, similar to the approach in [10]. For convenience, we provide a summary of the online feasibility test based on priority inversion budget analysis here.

Definition 10 [7]: Given a task set $\Gamma(\pi_k)$ scheduled on a uniprocessor π_k, *the maximum priority inversion budget of task* $\boldsymbol{\tau_h \in \Gamma(\pi_k)}$ *at time* $\boldsymbol{t}$, denoted by $\mathcal{B}_h^t$, is defined as the maximum amount of time for which all lower priority tasks $lp(\tau_h)$ are allowed to execute before τ_h finishes at t, while ensuring the schedulability of τ_h in the worst-case scenario.

In the analysis of the priority inversion budget for a task $\tau_h \in hp(\tau_{\text{test}})$, we consider its two adjacent jobs $\mathcal{J}_h^t$ and $\mathcal{J}_h^{t'}$, where job $\mathcal{J}_h^t$ is the last job of task τ_h released no later than time t and job $\mathcal{J}_h^{t'}$ is the first job of task τ_h released after time t. Depending on whether $\mathcal{J}_h^t$ is in the ready queue Q_t or not at time t, we consider two cases.

Algorithm 2 Online Feasibility Test

Input: Task set $\Gamma(\pi_k)$, $\overline{V}_i$ of $\tau_i \in \Gamma(\pi_k)$ calculated off-line with (19) and (20), test job $\mathcal{J}_{\text{test}}$, scheduling point t, ready queue Q_t, job $\mathcal{J}_h^t$ that is the last job of task $\tau_h \in hp(\tau_{\text{test}})$ released no later than time t.

Output: The feasibility *flag* of executing job $\mathcal{J}_{\text{test}}$ at time t, and the maximum feasible execution time E_{test}^t of job $\mathcal{J}_{\text{test}}$.

```
flag ← True
E^t_test ← C̃^t_test
for all τ_h ∈ hp(τ_test) do
    Calculate I_h(t, d^t_h) with equation (14)
    if J^t_h ∈ Q_t at time t then
        B̲^t_h ← d^t_h − t − C̃^t_h − I_h(t, d^t_h)
    else
        B̲^t_h ← d^t_h − t + V̄_h − I_h(t, d^t_h)
    end if
    if B^t_h > 0 then
        E^t_test ← min{E^t_test, B̄^t_h}
    else
        flag ← False
        E^t_test ← 0
        break
    end if
end for
return (flag, E^t_test)
```

Case 1 (Job $\mathcal{J}_h^t$ Is in Ready Queue Q_t at Time t): In this case, we can focus solely on the analysis of the job $\mathcal{J}_h^t$ of task τ_h at time t. Let d_h^t be the absolute deadline of job $\mathcal{J}_h^t$. Based on Definition 10, by analyzing the workload during the time interval $[t, d_h^t)$, the maximum priority inversion budget $\mathcal{B}_h^t$ of task τ_h at time t can be expressed as

$$\mathcal{B}_h^t \geq d_h^t - t - \tilde{C}_h^t - I_h\left(t, d_h^t\right) \tag{13}$$

where $\tilde{C}_h^t$ is the worst-case remaining execution time of job $\mathcal{J}_h^t$. $I_h(t, d_h^t)$ is the worst-case workload of tasks in $hp(\tau_h)$ during the time interval $[t, d_h^t)$, and it can be calculated by

$$I_h\left(t, d_h^t\right) = \sum_{\tau_j \in hp(\tau_h)} \tilde{C}_j^t + \sum_{\tau_j \in hp(\tau_h)} \left\lceil \frac{d_h^t - r_j^{t'}}{T_j} \right\rceil_0 C_j \tag{14}$$

where $\lceil x \rceil_0$ is the smallest non-negative integer greater than or equal to x, $\tilde{C}_j^t$ is the worst-case remaining execution time of task τ_j at time t, and $r_j^{t'}$ is the release time of the first job of task τ_j released after time t.

Case 2 (Job $\mathcal{J}_h^t$ Is Not in Ready Queue Q_t at Time t): In this scenario, the upcoming job $\mathcal{J}_h^{t'}$ of task τ_h released after time t should be analyzed. Let $r_h^{t'}$ and $d_h^{t'}$ be the release time and the absolute deadline of job $\mathcal{J}_h^{t'}$. Based on Definition 10, the maximum priority inversion budget $\mathcal{B}_h^t$ of task τ_h at time t can be expressed as

$$\mathcal{B}_h^t = \mathcal{B}_h\left(t, r_h^{t'}\right) + \mathcal{B}_h\left(r_h^{t'}, d_h^{t'}\right) \tag{15}$$

where $\mathcal{B}_h(t, r_h^{t'})$ and $\mathcal{B}_h(r_h^{t'}, d_h^{t'})$ are the maximum priority inversion budgets of task τ_h during time intervals $[t, r_h^{t'})$ and

$[r_h^{t'}, d_h^{t'})$, respectively. By analyzing the workload during the time interval $[t, r_h^{t'})$, $\mathcal{B}_h^t(t, r_h^{t'})$ can be expressed as

$$\mathcal{B}_h\left(t, r_h^{t'}\right) \geq r_h^{t'} - t - I_h\left(t, r_h^{t'}\right) \tag{16}$$

where $I_h(t, r_h^{t'})$ is the worst-case workload of tasks in $hp(\tau_h)$ during the time interval $[t, r_h^{t'})$. Since task τ_h is an implicit-deadline periodic task (i.e., $r_h^{t'} = d_h^t$), (16) can rewritten as

$$\mathcal{B}_h\left(t, r_h^{t'}\right) \geq d_h^t - t - I_h\left(t, d_h^t\right) \tag{17}$$

where $I_h(t, d_h^t)$ can be calculated with (14).

For the maximum priority inversion budget $\mathcal{B}_h(r_h^{t'}, d_h^{t'})$ of task τ_h during time interval $[r_h^{t'}, d_h^{t'})$, it can be expressed as

$$\mathcal{B}_h\left(r_h^{t'}, d_h^{t'}\right) \geq \overline{V}_h \tag{18}$$

where $\overline{V}_h$ is the maximum amount of time that τ_h can additionally have while meeting its deadline when there are no deferred executions of tasks in $hp(\tau_h)$ at the release time of task τ_h. According to [14], $\overline{V}_h$ can be calculated offline, and it can be expressed as

$$\overline{V}_h = \max\ \{\delta \mid W_h(\delta) \leq D_h\} \tag{19}$$

where D_h is the relative deadline of task τ_h, and $W_h(\delta)$ is the duration between a critical instant and the response completion of the corresponding request of task τ_h with extra execution time δ. According to [23], $W_h(\delta)$ can be derived by solving the following iterative formula:

$$W_h^{n+1}(\delta) = W_h^0(\delta) + \sum_{\tau_j \in hp(\tau_h)} \left\lceil \frac{W_h^n(\delta)}{T_j} \right\rceil C_j \tag{20}$$

where $W_h^0(\delta)$ is the initial value of $W_h(\delta)$, and it is set as $C_h + \delta$ by considering the WCET and the extra execution time of task τ_h. $W_h^n(\delta)$ is the value of $W_h(\delta)$ at the nth iteration.

By (15), (17), and (18), the maximum priority inversion budget $\mathcal{B}_h^t$ of task τ_h at time t can be expressed as

$$\mathcal{B}_h^t \geq d_h^t - t - I_h\left(t, d_h^t\right) + \overline{V}_h. \tag{21}$$

Based on the analysis of the two cases mentioned above, the maximum priority inversion budget $\mathcal{B}_h^t$ of task τ_h at time t is bounded by (13) and (21). Hence, a lower bound of $\mathcal{B}_h^t$, denoted by $\overline{\mathcal{B}_h^t}$, can be calculated by

$$\overline{\mathcal{B}_h^t} = \begin{cases} d_h^t - t - \tilde{C}_h^t - I_h\left(t, d_h^t\right) & \mathcal{J}_h^t \in Q_t \text{ at time } t \\ d_h^t - t + \overline{V}_h - I_h\left(t, d_h^t\right) & \mathcal{J}_h^t \notin Q_t \text{ at time } t. \end{cases} \tag{22}$$

From (22), for a task τ_{test}, if $\overline{\mathcal{B}_h^t}$ is greater than zero for each task $\tau_h \in hp(\tau_{\text{test}})$, it is feasible to execute task τ_{test} at time t by the priority inversion. Thus, we can obtain the following theorem.

Theorem 2 [10]: For a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k under the RM policy, if the job $\mathcal{J}_{\text{test}}$ of task $\tau_{\text{test}} \in \Gamma(\pi_k)$ is executed with execution time E_{test}^t at time t when $E_{\text{test}}^t \leq \overline{\mathcal{B}_h^t}$ for all tasks $\tau_h \in hp(\tau_{\text{test}})$ and $\overline{\mathcal{B}_h^t}$ is calculated with (22), then the task set $\Gamma(\pi_k)$ is also schedulable.

Proof: For the detailed proof, please refer to [10]. ■

As shown in Algorithm 2, it is the priority inversion budget-based online feasibility test algorithm written in pseudocode. In this algorithm, *flag* indicates whether $\mathcal{J}_{\text{test}}$ can be executed at time t and E_{test}^t denotes its maximum feasible execution time at time t. First, *flag* and E_{test}^t are initialized to True and $\tilde{C}_{\text{test}}^t$, respectively (lines 1 and 2). Then, the feasibility of executing the job $\mathcal{J}_{\text{test}}$ at time t is checked by calculating the lower bound of the maximum priority inversion budget for each task $\tau_h \in hp(\tau_{\text{test}})$. The upper bound on the workload of each task in $hp(\tau_h)$ during the interval $[t, d_h^t)$ is calculated by (14) (line 4). The lower bound of the maximum priority inversion budget for task τ_h is calculated by (22) (lines 5–9). If $\overline{\mathcal{B}_h^t} > 0$, the job $\mathcal{J}_{\text{test}}$ can be executed at t with execution time $\overline{\mathcal{B}_h^t}$ while ensuring the schedulability of task τ_h, and E_{test}^t is updated to $\min\{E_{\text{test}}^t, \overline{\mathcal{B}_h^t}\}$ (lines 10 and 11). If there exists a task $\tau_h \in hp(\tau_{\text{test}})$ whose $\overline{\mathcal{B}_h^t}$ is not greater than 0, it is not feasible to execute $\mathcal{J}_{\text{test}}$ at time t (lines 12–16); otherwise, it is feasible.

Lemma 3: Let $\Gamma(\pi_k)$ be the set of tasks scheduled on the uniprocessor π_k. The execution feasibility of a job $\mathcal{J}_{\text{test}}$ of task $\tau_{\text{test}} \in \Gamma(\pi_k)$ at time t can be obtained with the computational complexity $\mathcal{O}(|hp(\tau_{\text{test}})|^2)$ with Algorithm 2, where $hp(\tau_{\text{test}})$ is the set of higher priority tasks of τ_{test} in task set $\Gamma(\pi_k)$ and $|hp(\tau_{\text{test}})|$ is the number of tasks in $hp(\tau_{\text{test}})$.

Proof: From Algorithm 2, it is evident that the computational complexity of the priority inversion budget-based online feasibility test for the job $\mathcal{J}_{\text{test}}$ depends mainly on the number of tasks with higher priority than τ_{test} (i.e., line 3) and the complexity to calculate $I_h(t, d_h^t)$ with (14) (i.e., line 4). The number of tasks with higher priority than task τ_{test} is $|hp(\tau_{\text{test}})|$. The complexity to calculate $I_h(t, d_h^t)$ with (14) is $\mathcal{O}(|hp(\tau_h)|)$ for each task $\tau_h \in hp(\tau_{\text{test}})$. Since $|hp(\tau_h)| \leq |hp(\tau_{\text{test}})|$ for each task $\tau_h \in hp(\tau_{\text{test}})$, we can obtain that the complexity of the feasibility test with Algorithm 2 for the job $\mathcal{J}_{\text{test}}$ is $\mathcal{O}(|hp(\tau_{\text{test}})|^2)$. ■

Theorem 3: For a schedulable task set $\Gamma(\pi_k)$ on a uniprocessor π_k under the RM policy, it is also schedulable with the multimode security-aware scheduling strategy given in Algorithm 1.

Proof: According to lines 12–21 in Algorithm 1, for each system mode, any job of tasks in $\Gamma(\pi_k)$ is executed with a priority inversion or based on the priority assigned with the RM policy. According to Theorem 2, for each system mode, the schedulability of $\Gamma(\pi_k)$ can be ensured when some jobs are executed with priority inversion based on the online feasibility test given in Algorithm 2, since the task set $\Gamma(\pi_k)$ is schedulable under the RM policy. Thus, mode schedulability can be ensured for all system modes. Moreover, when an MCR associated with a mode translation arrives, the system will immediately enter the new mode and perform the enforcement by calling Algorithm 1, and thus mode transition schedulability can also be ensured. Therefore, we can conclude that the task set $\Gamma(\pi_k)$ is also schedulable under the multimode security-aware scheduling strategy. ■

Theorem 4: Let $\Gamma(\pi_k)$ be the task set scheduled on uniprocessor π_k with the multimode security-aware scheduling policy given in Algorithm 1. The computational complexity of

Algorithm 1 is $\mathcal{O}(|\Gamma(\pi_k)|^2)$, where $|\Gamma(\pi_k)|$ is the number of tasks in the task set $\Gamma(\pi_k)$.

Proof: From Algorithm 1, it is evident that the computational complexity of the multimode security-aware scheduling algorithm primarily relies on the computational complexity of the online feasibility test based on the priority inversion budget analysis given in Algorithm 2, and the feasibility test is conducted no more than once for each scheduling point (lines 12–21). By referring to Lemma 3, it can be deduced that the computational complexity of Algorithm 1 is $\mathcal{O}(|\Gamma(\pi_k)|^2)$, which is polynomial complexity. ∎

Algorithm 3 Security-Aware Task Partitioning

Input: Task set $\Gamma = \Gamma^{\mathsf{v}} \cup \Gamma^{\mathsf{t}} \cup \Gamma^{\mathsf{u}}$, multiprocessor platform $\Pi = \{\pi_1,...,\pi_P\}$.
Output: Task partitions $\{\Gamma(\pi_1), \Gamma(\pi_2), ..., \Gamma(\pi_P)\}$.
1: $\Gamma(\pi_k) \leftarrow \emptyset$, for all $k = 1, ..., P$.
2: **for** all $\tau_i^{\mathsf{v}} \in \Gamma^{\mathsf{v}}$ in descending order of U_i^{v} **do**
3:   *flag* ← False
4:   **for** all $\pi_k \in \Pi$ in increasing order of $U_{\Gamma(\pi_k)}$ **do**
5:     **if** it is feasible for task set $\Gamma(\pi_k) \cup \{\tau_i^{\mathsf{v}}\}$ **then**
6:       $\Gamma^{\mathsf{v}} \leftarrow \Gamma^{\mathsf{v}} \setminus \tau_i^{\mathsf{v}}$
7:       $\Gamma(\pi_k) \leftarrow \Gamma(\pi_k) \cup \{\tau_i^{\mathsf{v}}\}$
8:       *flag* ← True
9:       **break**
10:     **end if**
11:   **end for**
12:   **if** *flag* = False **then**
13:     **return** FAILURE
14:   **end if**
15: **end for**
16: **for** all $\tau_i^{\mathsf{t}} \in \Gamma^{\mathsf{t}}$ in descending order of U_i^{t} **do**
17:   *flag* ← False
18:   **for** all $\pi_k \in \Pi$ in increasing order of $U_{\Gamma(\pi_k)}$ **do**
19:     **if** it is feasible for task set $\Gamma(\pi_k) \cup \{\tau_i^{\mathsf{t}}\}$ **then**
20:       $\Gamma^{\mathsf{t}} \leftarrow \Gamma^{\mathsf{t}} \setminus \tau_i^{\mathsf{t}}$
21:       $\Gamma(\pi_k) \leftarrow \Gamma(\pi_k) \cup \{\tau_i^{\mathsf{t}}\}$
22:       *flag* ← True
23:       **break**
24:     **end if**
25:   **end for**
26:   **if** *flag* = False **then**
27:     **return** FAILURE
28:   **end if**
29: **end for**
30: **for** all $\tau_i^{\mathsf{u}} \in \Gamma^{\mathsf{u}}$ in descending order of U_i^{u} **do**
31:   *flag* ← False
32:   **for** all $\pi_k \in \Pi$ in increasing order of $U_{\Gamma(\pi_k)}$ **do**
33:     **if** it is feasible for task set $\Gamma(\pi_k) \cup \{\tau_i^{\mathsf{u}}\}$ **then**
34:       $\Gamma^{\mathsf{u}} \leftarrow \Gamma^{\mathsf{u}} \setminus \tau_i^{\mathsf{u}}$
35:       $\Gamma(\pi_k) \leftarrow \Gamma(\pi_k) \cup \{\tau_i^{\mathsf{u}}\}$
36:       *flag* ← True
37:       **break**
38:     **end if**
39:   **end for**
40:   **if** *flag* = False **then**
41:     **return** FAILURE
42:   **end if**
43: **end for**
44: **return** SUCCESS

E. Security-Aware Task Partitioning

In this section, we present a partitioning algorithm to assign a set of mixed-trust real-time tasks Γ to P identical, unit-capacity processors Π. The objective of this algorithm is to balance the mixed-trust task workloads across processors while ensuring system schedulability, such that the system protection capability can be maximized under the schedulability constraint. This is achieved with a mixed-trust WF decreasing heuristic strategy, which sequentially selects victim tasks, trusted tasks, and untrusted tasks to assign based on the WF decreasing heuristic.

Algorithm 3 is our partitioning algorithm, written in pseudocode. Here, the set of tasks assigned to processor π_k is identified by $\Gamma(\pi_k)$. The algorithm starts by initializing $\Gamma(\pi_k)$ to null (line 1). Then, it tries to assign suitable processors to victim tasks (lines 4–15), trusted tasks (lines 16–29), and untrusted tasks (lines 30–43) in sequence. For each type of tasks, tasks are verified in descending utilization order (lines 4, 16, and 30). In this order, each task is tried on each of the processors in Π, ordered in increasing utilization. For a task, if no feasible processor is found, the algorithm aborts with a failure (lines 13, 27, and 41). If all tasks are successfully assigned, the algorithm reports success (line 44).

Theorem 5: Given an implicit-deadline periodic real-time task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$, if it is successfully partitioned by Algorithm 3 on P processors $\Pi = \{\pi_1, \pi_2, \ldots, \pi_P\}$ where the feasibility test of each processor is performed based on the schedulability condition of the RM policy, then all tasks in Γ can meet their deadlines under the multimode security-aware scheduling policy given in Algorithm 1.

Proof: We consider any processor $\pi_k \in \Pi$, and the task set $\Gamma(\pi_k)$ assigned to π_k. According to lines 5–10, 19-24, and 33–38 in Algorithm 3, the task set $\Gamma(\pi_k)$ is feasible on processor π_k under the RM scheduling policy. Based on Theorem 3, task set $\Gamma(\pi_k)$ is also feasible under the scheduling strategy given in Algorithm 1. This implies that the task set on each local processor can be successfully scheduled. Consequently, we can conclude that the resulting task allocation obtained by Algorithm 3 always guarantees that all tasks in Γ can meet the deadlines under the multimode security-aware scheduling policy given in Algorithm 1 when task set Γ is successfully partitioned by Algorithm 3. ∎

Example 2: Consider the task set Γ given in Table I again. We consider that the task set Γ is allocated to four processors $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ with the security-aware task partitioning algorithm given in Algorithm 3, and the tasks allocated to each processor are scheduled with the multimode security-aware scheduling algorithm given in Algorithm 1. From the SAS simulation for the task set Γ given in Fig. 2, we can see that all tasks are schedulable and that all untrusted jobs are executed outside AEWs of all victim tasks to protect all victim jobs from being attacked by untrusted tasks. We can see that the accumulative AEW within time interval $[0, 20)$ is $6 + 3 = 9$, and hence the AEW ratio within time interval $[0, 20)$ is $9/20 = 0.45$. Since there is no untrusted task execution within AEWs in time interval $[0, 20)$, we can obtain that the AEW untrusted execution time ratio within time interval $[0, 20)$ is 0. Therefore, MM-SARTS can provide better protection than

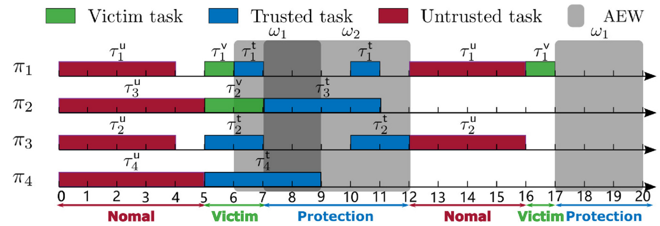

Fig. 2. SAS simulation for task set Γ under MM-SARTS.

SchedGuard++ by effectively reducing the AEW ratio and the AEW untrusted execution time ratio with security-aware task partitioning and multimode security-aware scheduling.

V. Evaluation

We conducted experimental evaluations to assess the performance of our multimode security-aware multiprocessor real-time scheduling method using synthetically generated workloads based on automotive benchmark [24]. In the experiment, we implemented the following six algorithms.

1) *RM-FF:* RM scheduling combining with the first-fit (FF) bin-packing heuristic algorithm.
2) *RM-NF:* RM scheduling combining with the next-fit (NF) bin-packing heuristic algorithm.
3) *RM-BF:* RM scheduling combining with the best-fit (BF) bin-packing heuristic algorithm.
4) *RM-WF:* RM scheduling combining with the WF bin-packing heuristic algorithm.
5) *SchedGuard++:* The security-aware multiprocessor real-time scheduling method from [11].
6) *MM-SARTS:* Our multimode security-aware multiprocessor real-time scheduling method.

Objectives: Our evaluation has two primary goals: 1) to compare the attack defense performance of our multimode security-aware multiprocessor scheduling approach with existing scheduling methods in terms of AEW ratio, AEW untrusted execution time ratio and ScheduLeak attack defense effect and 2) to assess the overhead of different scheduling methods by measuring the scheduler CPU time consumption with time.process_time_ns() method in the Python time module.

A. Evaluation Setup

The periods of all tasks are automotive specific semi-harmonic, and they are drawn at random from the set {1, 2, 5, 10, 20, 50, 100, 200, 1000} with an associated appearance probability given in [24]. Task utilization is determined with the UUniFast approach from [25] and task priorities are assigned according to the RM scheduling policy. We define the normalized utilization $U_{\text{nor}}(\Gamma)$ of a task set Γ deployed on the multiprocessor platform $\Pi = \{\pi_1, \ldots, \pi_P\}$ to be

$$U_{\text{nor}}(\Gamma) = \frac{U_\Gamma}{P} \tag{23}$$

where U_Γ is the total utilization of the task set Γ, and P is the processor number. To ensure the generated task set's utilization within a specific narrow range around the desired normalized utilization, with the above parameters, we generated the tasks one at a time until the system utilization met the condition

$$U^*_{\text{nor}} - 0.005 \le U_{\text{nor}}(\Gamma) \le U^*_{\text{nor}} + 0.005 \tag{24}$$

where U^*_{nor} ranges from 0.05 to 0.95 with step size of 0.05.

We consider that there are four processors (i.e., $P = 4$) in the system. For each value of U^*_{nor}, we generate 1000 task sets. For each task set, there are 20–30 tasks. The experimental results are averaged over these 1000 task sets.

Following the experimental setting in [11], 40% of the tasks within the task set are chosen at random to be considered as trusted tasks. 50% of the trusted tasks are randomly selected as victim tasks while excluding the lowest priority task in the task set. The AEWs of the victim tasks are determined as a percentage from the set {10, 30, 50} of the period.

To validate the performance of MM-SARTS against the schedule-based attack, we evaluated the defensive capabilities of different scheduling approaches against the ScheduLeak attack [2]. ScheduLeak is a common schedule-based attack that utilizes a system timer to measure and reconstruct the valid execution intervals of the attacker task with a lower priority than the victim task. In our experiments, a ScheduLeak attacker task is chosen at random from the untrusted tasks in the task set. It is important to note that MM-SARTS can be applied to the system with multiple attacker tasks. This is because in MM-SARTS, every untrusted task is viewed as a potential attacker task in the security-aware scheduling process, and thus MM-SARTS can offer protection against all untrusted tasks rather than focusing on a specific untrusted task.

The experiments were conducted on a desktop computer equipped with an AMD Ryzen 7 5800H CPU running at 3.20 GHz with 8 cores, featuring 16 GB of physical RAM, and operating on a Linux kernel version 5.15.0-88-generic.

B. Results

AEW Ratio: Fig. 3 illustrates the AEW ratio within a hyperperiod versus the utilization of the task set of different algorithms for the systems with different AEW sizes. Fig. 3 shows that RM-WF has a lower AEW ratio relative to the other nonsecurity-aware RM scheduling approaches. The possible reason for this is that RM-WF can effectively balance the workload across different processors, thus fully utilizing the system's parallel processing capabilities to reduce the AEW ratio. We can also observe that *SchedGuard++* performs worse than all nonsecurity-aware RM scheduling methods in most cases, especially for the system with short AEWs. The main reason for this is that once a victim task finishes, *SchedGuard++* blocks all other processors, including processors running trusted tasks. This decreases the chance that victim tasks execute in parallel, thereby reducing the part of AEWs that overlaps between different processors.

Our main result is that MM-SARTS *consistently outperforms* all existing scheduling algorithms. The performance gap tends to widen as the utilization decreases; the reason is that, as the utilization decreases, there are more opportunities for MM-SARTS to reduce the AEW ratio by increasing the

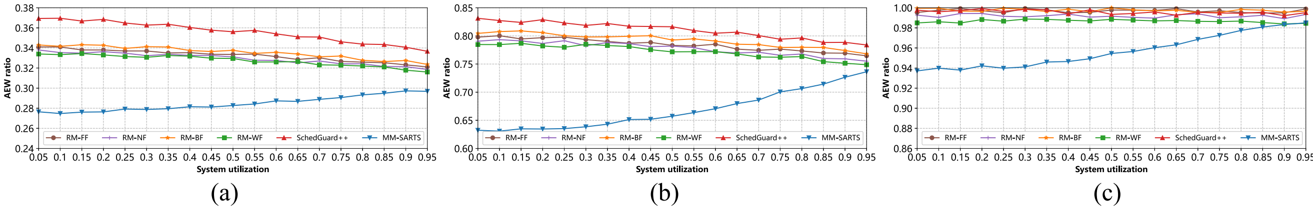

Fig. 3. AEW ratio for different algorithms. (a) AEW is 10% of victim task period. (b) AEW is 30% of victim task period. (c) AEW is 50% of victim task period.

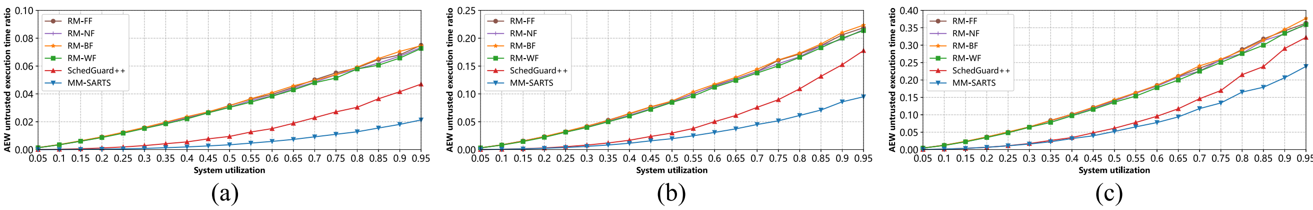

Fig. 4. AEW untrusted execution time ratio for different algorithms. (a) AEW is 10% of victim task period. (b) AEW is 30% of victim task period. (c) AEW is 50% of victim task period.

AEW overlap with security-aware task-to-processor packing and multimode security-aware scheduling. Specifically, for the system with an AEW size of 10%, when the system utilization is 0.60, MM-SARTS demonstrates an enhancement of approximately 18.8% over *SchedGuard++*, and a boost of around 13.2% compared to the best nonsecurity-aware RM scheduling method RM-WF [see Fig. 3(a)]. From Fig. 3(a), we can also find that the improvement of MM-SARTS in the AEW ratio is at its lowest point when the system utilization is 0.95. Even in this case, MM-SARTS still achieves an improvement of 11.8% over *SchedGuard++*, and a gain of 7.5% compared to the RM-WF scheduling method.

AEW Untrusted Execution Time Ratio: Fig. 4 illustrates the AEW untrusted execution time ratio within a hyperperiod versus the system utilization of different algorithms for the systems with different AEW sizes. It clearly shows that RM-WF has a lower AEW untrusted execution time ratio compared to the other three RM scheduling approaches. We can observe that as the AEW size and system utilization increase, the AEW untrusted execution time ratios for all algorithms also increase. The reason is that, as the AEW size and system utilization increase, the AEW ratio and the untrusted task workload tend to increase. Note that when the system utilization is 0.05, for all AEW settings, the AEW untrusted execution time ratios of all methods tend to approach zero, though there is still a portion of task sets with a ratio greater than zero. Moreover, it is notable that *SchedGuard++* outperforms all nonsecurity-aware RM-WF scheduling methods by effectively preventing the execution of untrusted tasks during the specified AEW.

It is not surprising that MM-SARTS *consistently performs better* than *SchedGuard++* in all scenarios. This is primarily due to MM-SARTS effectively reducing the AEW untrusted execution time ratio in a multiprocessor real-time system with multiple victims by distributing the mixed-trust task workload evenly across different processors and strategically scheduling mixed-trust tasks on each processor based on the system mode. Moreover, the enhancement in the AEW untrusted execution time ratio of MM-SARTS tends to rise as the system utilization increases for all AEW sizes. Specifically, in a system with an AEW size of 10%, when the system utilization is 0.6, MM-SARTS shows a 62.8% improvement compared to *SchedGuard++* and an 86.8% improvement compared to the RM-WF scheduling method [see Fig. 4(a)].

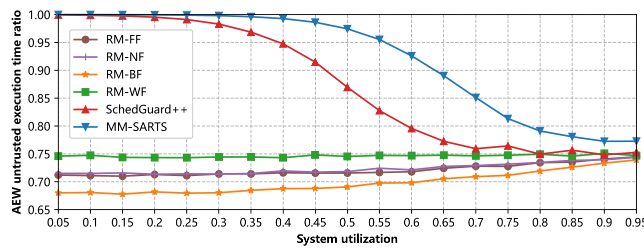

Fig. 5. Defense success rate of different algorithms.

ScheduLeak Attack Defense Effect: Fig. 5 illustrates the defense success rate against ScheduLeak versus the task set utilization of different algorithms. Here, the defense success rate is calculated as the failure rate of ScheduLeak to infer accurate parameters of the victim task. From Fig. 5, we can see that RM-WF has a higher defense success rate relative to the other three RM scheduling approaches. We also can see that both *SchedGuard++* and MM-SARTS exhibit superior defense capabilities against attacks compared to nonsecurity-aware RM-WF scheduling methods. This superiority is attributed to the ability of *SchedGuard++* and MM-SARTS to prevent the attacker task from running after the victim task completes, creating a false impression for the attacker about the victim's execution time. Consequently, the attacker is misled into launching an attack at an incorrect time based on inaccurate timing information. Additionally, it can be observed that MM-SARTS *consistently surpasses SchedGuard++*, as MM-SARTS enhances defense effectiveness through security-aware task-to-processor allocation and multimode security-aware scheduling based on online feasibility test. Specifically, when the system utilization is 0.6, MM-SARTS shows a 16.3% improvement compared to *SchedGuard++*.

Scheduler Overhead: Fig. 6 illustrates a comparison of the average online scheduling time within ten hyperperiods versus

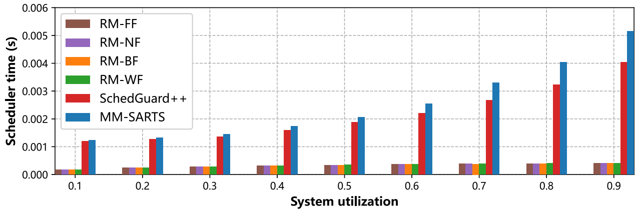

Fig. 6. Average online scheduling time of different algorithms.

system utilization for various algorithms. It is evident that the nonsecurity-aware RM scheduling methods have the least scheduler overhead, as they do not require online feasibility test for the priority inversion. We also can see that the overhead of MM-SARTS is slightly higher than that of *SchedGuard++*. The main reason for this is that MM-SARTS explores more opportunities for priority inversion to enhance protection performance, resulting in more online feasibility tests.

VI. Conclusion

We have proposed MM-SARTS, a multimode security-aware real-time scheduling technique against schedule-based attacks in fixed-priority real-time systems on multiprocessors. MM-SARTS works by distinctively scheduling mixed-trust tasks with an online priority inversion feasibility test according to system modes to minimize the AEW ratio and the AEW untrusted execution time ratio. In particular, we introduce the protection window for multiple victims on multiprocessors to avoid the protection degradation due to the excessive blocking of untrusted tasks by analyzing the system protection capability limit under the system schedulability constraint. To maximize the protection capability of the multimode security-aware scheduling strategy on the multiprocessor platform, we also propose a security-aware task-to-processor packing algorithm to balance the workloads of mixed-trust tasks across different processors. Our evaluation shows that MM-SARTS surpasses the existing security-aware scheduling method for fixed-priority real-time systems on multiprocessors in terms of the AEW ratio, the AEW untrusted execution time ratio, and the attack defense capability.

References

[1] D. Trilla et al., "Cache side-channel attacks and time-predictability in high-performance critical real-time systems," in *Proc. 55th Annu. Design Autom. Conf.*, 2018, pp. 1–6.

[2] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Montreal, QC, Canada, 2019, pp. 90–102.

[3] S. Bi and Y. J. Zhang, "False-data injection attack to control real-time price in electricity market," in *Proc. GLOBECOM*, 2013, pp. 772–777.

[4] M. Luo et al., "Stealthy tracking of autonomous vehicles with cache side channels," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 859–876.

[5] J. Chen et al., "SchedGuard++: Protecting against schedule leaks using Linux containers," in *Proc. IEEE 27th Real-Time Embed. Technol. Appl. Symp.*, 2021, pp. 14–26.

[6] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks," in *Proc. IEEE Real-Time Embedd. Technol. Appl. Symp. (RTAS)*, 2019, pp. 103–116.

[7] M.-K. Yoon, S. Mohan, C.-Y. Chen, and L. Sha, "TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems," in *Proc. RTAS*, 2016, pp. 1–12.

[8] C.-Y. Chen, M. Hasan, A. Ghassami, S. Mohan, and N. Kiyavash, "REORDER: Securing dynamic-priority real-time systems using schedule obfuscation," 2018, *arXiv:1806.01393*.

[9] J. Ren et al., "REORDER++: Enhanced randomized real-time scheduling strategy against side-channel attacks," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 6, pp. 3253–3266, Nov./Dec. 2023.

[10] J. Ren et al., "Protection window based security-aware scheduling against schedule-based attacks," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 5s, pp. 1–22, 2023.

[11] J. Chen et al., "SchedGuard++: Protecting against schedule leaks using Linux containers on multi-core processors," *ACM Trans. Cyber-Phys. Syst.*, vol. 7, no. 1, pp. 1–25, 2023.

[12] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba, "Period adaptation for continuous security monitoring in multicore real-time systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2020, pp. 430–435.

[13] J. Son et al., "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in MLS systems," in *Proc. IEEE Inf. Assur. Workshop*, 2006, pp. 361–368.

[14] M.-K. Yoon, J.-E. Kim, R. Bradford, and Z. Shao, "TaskShuffler++: Real-time schedule randomization for reducing worst-case vulnerability to timing inference attacks," 2019, *arXiv:1911.07726*.

[15] K. Krüger, M. Völp, and G. Fohler, "Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems," in *Proc. 30th ECRTS*, 2018, pp. 1–22.

[16] K. Krüger et al., "Randomization as mitigation of directed timing inference based attacks on time-triggered real-time systems with task replication," *Leibniz Trans. Embed. Syst.*, vol. 7, no. 1, pp. 01:1–01:29, 2021.

[17] M.-K. Yoon, J.-E. Kim, R. Bradford, and Z. Shao, "TimeDice: Schedulability-preserving priority inversion for mitigating covert timing channels between real-time partitions," in *Proc. 52nd DSN*, 2022, pp. 453–465.

[18] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, 2008, pp. 44–55.

[19] S. Mohan, M. K. Yoon, R. Pellizzoni, and R. B. Bobba, "Real-time systems security through scheduler constraints," in *Proc. 26th ECRTS*, 2014, pp. 129–140.

[20] R. Pellizzoni et al., "A generalized model for preventing information leakage in hard real-time systems," in *Proc. 21st RTAS*, 2015, pp. 271–282.

[21] *Road Vehicles Open Interface for Embedded Automotive Applications Part 3: OSEK/VDX Operating System*, ISO Standard 17356-3, 2005.

[22] "Specification of operating system," AUTOSAR Consortium, Hörgertshausen, Germany, Rep. CP R22-11, 2022.

[23] C. L. Liu et al., "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[24] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proc. WATERS*, 2015, pp. 1–6.

[25] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, 2005.