A Secure Resilient Real-Time Recovery Model, Scheduler, and Analysis

Abdullah Al Arafat*, Sudharsan Vaidhun†, Bryan C. Ward‡, Zhishan Guo*
*Department of Computer Science, North Carolina State University
†Department of Electrical and Computer Engineering, University of Central Florida
‡Department of Computer Science, Vanderbilt University

***Abstract*—Real-time and embedded systems are increasingly being applied in the command and control of safety- and mission-critical applications such as autonomous vehicles and critical infrastructure. Meanwhile, to enable new capabilities, we are witnessing a rapid growth in the complexity and connectivity of such devices. Unfortunately, such designs often introduce new attack vectors, necessitating inventions that provide stronger security and resilience. This paper presents a secure and resilient scheduling technique for hard real-time applications. Specifically, this approach builds upon the well-known mixed-criticality scheduling framework and demonstrates a new dimension of criticality: security criticality. In the presented model, low-security-criticality workloads are dropped in the event of a malicious event, both to minimize the attack surface, as well as enable the timely scheduling of both a recovery task and the re-execution of the victim task. This paper demonstrates how the existing mixed-criticality scheduling approaches are overly pessimistic in light of this model, and presents a new scheduling algorithm for it. The performance of the presented algorithm and analysis is evaluated through schedulability experiments.**

I. Introduction

Real-time and embedded systems are being employed across society to monitor and control increasingly complex cyber-physical systems. For example, modern automobiles have dozens of onboard computers to control the engine, transmission, braking, driver-assist features, and infotainment systems. In industrial-control applications, the Industrial Internet of Things (IIoT) promises greater efficiencies through increased communication, coordination, and autonomy of industrial processes ranging from power systems to manufacturing.

As a society, we are increasingly reliant upon such systems and enjoy the capabilities and efficiencies such interconnected systems offer. However, the complexity of these systems increases the attack surface, and their increasing connectivity makes vulnerabilities even more accessible to attackers. Furthermore, embedded systems are often not developed with the same cyber-security scrutiny that is common in general-purpose systems. For example, the Mirai botnet [2] exploited the unchanged universal factory-default password to co-opt webcams into a powerful botnet. While many attacks can be detected and/or prevented via known defensive techniques, it is critical that a system is able to respond and recover from such threats while preserving real-time constraints.

Common cyber-security defenses are often eschewed in real-time and embedded systems. For example, address-space layout randomization (ASLR) is employed ubiquitously across general-purpose computing systems and is enabled by default in Windows, Mac OSX, and Linux. But randomization-based defenses are often avoided in real-time systems for predictability reasons as they can significantly increase worst-case performance [7], [9].

The most common class of vulnerabilities are *memory-corruption* vulnerabilities, which are bugs that an attacker can exploit to corrupt regions of memory. Microsoft and Google have reported such vulnerabilities account for approximately 70% of vulnerabilities in their codebases [19], [22]. While there are techniques to eliminate such vulnerabilities, they are expensive (Softbound [18] has overheads over 100%), or impractical (rewrite all code in a memory-safe language such as Rust). Therefore, most runtime defenses protect against memory-corruption-based attacks by seeking to prevent exploitation by crashing the process. For example, control-flow integrity (CFI) [1], [24] performs checks at control-flow transitions to ensure valid branch targets, and crashes the process upon invalid control flow. Importantly, runtime defenses are *integral* to the protected task, that is, they are executed within the protected process, not in a separate process as in monitoring-based security approaches [10], [11], [12], [13]. We note that runtime defenses are designed to prevent exploitation, while monitoring-based approaches detect anomalies and evidence that the system has been compromised. Therefore, monitoring-based security approaches are orthogonal to the runtime defenses we consider, but are outside the scope of this work.

Processes in real-time and embedded systems often control physical devices, and hence cannot simply crash to prevent an attack—such an approach could itself compromise the system. Instead, in such applications, computation could be restarted to maintain continuous safe control of the physical process. However, restarting a real-time job may significantly impact its ability to complete before its deadline, and may introduce additional demand that may in turn compromise the temporal integrity of other tasks in the system, if not properly mitigated.

These observations motivate the need for new task models, scheduling algorithms, and analysis to enable resilience to cyber attacks, *i.e.*, the ability to maintain some safe level of operation while recovering from an attack. While there may be simple or naïve means of supporting such behavior in existing scheduling and analysis frameworks, fully maximizing the platform utilization while enabling such resilience requires fundamentally new models, algorithms, and analysis.

Mixed criticality. This problem has several important commonalities with mixed-criticality (MC) scheduling, specifi-

cally, the ability to operate in a degraded mode of execution. Critically, however, MC scheduling models have principally been developed to handle one aberrant behavior—temporal overruns—not security incidence. However, Burns has recently argued that work on MC systems should be generalized to multi-mode systems [6]. This work is an exemplar of this argument, and we demonstrate a multi-mode system in which mode switches are triggered by security events rather than timing overruns.

There are several important *similarities and differences* between the standard Vestal-model [23] for MC scheduling and the needs of a resilient real-time recovery model. For example, when a security event is detected, it is useful to shed less-critical workloads to ensure the continued correct operation of high-criticality work. Shedding work is especially useful for security as it can also reduce the attack surface of the system. There are, however, several important differences. First, when a defense prevents an attack it crashes the process, requiring re-execution of the job and additional processing time. Another key difference with security criticality is that memory-corruption attacks are most likely to target only a single task, not all high-criticality tasks simultaneously. Memory-corruption attacks target vulnerabilities in code, and because different tasks have different code, they are not likely to be vulnerable to the same exploit payloads.

As a result, adapting existing MC system analysis results will be too pessimistic. In addition, in an MC environment, the system often returns to normal mode when a transient overload condition subsides. In contrast, returning to a normal mode of execution after detection of a cyber threat may require additional recovery processing for computations such as (i) adding the malicious input to a blocklist to ensure the re-executed task will not be attacked, [17] (ii) forensic analysis, (iii) human-operator communication, and/or (iv) other actions to harden the security posture of the system, such as substituting binaries with stronger-defended ones, *etc.* Such additional computation time must also be modeled and analyzed. Notably, shedding less-critical workload, with the proper analysis, frees computation time to enable such recovery processing without affecting the utilization of the normal mode.

Related Works. Previous work has studied co-scheduling security monitor tasks [10], [11], [12], [13] with real-time tasks in fixed-priority partitioned multi-core systems with/without allowing migration of monitor tasks. These papers assume that the security tasks monitor security events and potentially detect the attacks (*i.e.*, works as intrusion detection system (IDS)). However IDS does not *stop* attacks, they merely attempt to detect malicious activity, while run-time defenses (*e.g.*, CFI [8], [24], [27], data flow integrity (DFI) [4]) prevent attacks from succeeding by crashing the process. Note that, unlike IDS, runtime defenses are integral to the task itself, and are *not independently scheduled.* Therefore, detection using runtime defenses is real-time and does not have any scheduling overhead. In SR^3, tasks are instrumented with a runtime defense instead of IDS.

Contributions. Based on these observations, we present the first resilient recovery scheduling model and analysis for secure real-time systems. We identify that temporal criticality and security criticality are orthogonal dimensions of criticality and that by designing a system of differing security criticalities enables both efficient recovery after an attack, as well as the minimization of the attack surface in the presence of a cyber threat. We make the following contributions:

- We propose SR^3, a secure and resilient real-time recovery task model that can recover from an attack at runtime while maintaining the real-time correctness of high-security-critical tasks.
- We develop a scheduling algorithm for the presented task model using earliest-first deadline (EDF) with modified virtual deadlines for security-critical tasks.
- We conduct schedulability evaluations that demonstrate the effectiveness of our scheduling algorithm over adapted existing scheduling schemes.

II. Model and Problem

A. Threat Model

We assume a threat model consistent with other prior works on run-time defenses [8], [24], [27]. Specifically, we assume a write-what-where vulnerability that an attacker can leverage to corrupt code pointers[1] to hijack control flow to attacker-specified location(s). Significant research has shown that even such simple and common vulnerabilities can be exploited using return-oriented programming (ROP) [21] or other attack techniques (*e.g.*, [26]) to completely hijack control flow and implement Turing-complete attacker-controlled logic. This is a very common and powerful threat model.

We assume the system is instrumented with a real-time runtime defense such as control-flow integrity (CFI) [8], [24], [27], [20], data-flow integrity [4], or an address-randomization defense [7], [9]. Notably, all of these defense techniques prevent further exploitation of a task by crashing the process.

Attacks on the scheduler or RTOS itself are outside the scope of our threat model. We note, however, that the scheduler and RTOS can be made trustworthy if using a verified RTOS, (*e.g.*, seL4 [15]), or by using a trusted execution environment (*e.g.*, ARM TrustZone) [25]. Notably, however, attacks related to the *confidentiality* (*e.g.*, side-channel attacks) of the workloads are out of the scope of this work. We also note that IDS as additional security tasks scheduled with regular workloads [10], [11], [12], [13] are outside the scope and, in fact, these models are orthogonal to SR^3.

B. System Model

Let $\tau' = \{\tau_1, \tau_2, ..., \tau_n\}$ be a set of n sporadic and implicit-deadline tasks scheduled on a uniprocessor. Each task τ_i can be represented by three tuple $\{C_i, T_i, \varsigma_i\}$, where C_i is the worst-case execution time (WCET), T_i is the minimal inter-arrival separation as well as the relative deadline (*i.e.*, $D_i = T_i$) of the task instances (jobs). We assume that τ_i is instrumented with runtime security defense(s), and that their overheads are

[1]A code pointer is any address stored in a data section that points to executable code. Return addresses on the stack are a frequent attacker targets.

TABLE I: Tasks of differing temporal and security criticalities.

		Temporal Criticality	
		High	Low
Security Criticality	High	Safety-critical Control Processing	Encryption key management software, or IDS
	Low	Processing non-mission-critical sensor inputs	Infotainment

included within C_i. We assume each task can potentially release an infinite sequence of jobs. Let $\varsigma_i \in \{0, 1\}$ denote whether task τ_i is of high or low security criticality. We use $\tau_\varsigma = \{\tau_i | \varsigma_i = 1\}$ and $\tau_\zeta = \{\tau_i | \varsigma_i = 0\}$ to denote the set of high-security-criticality (HI-security) tasks and low-security-criticality (LO-security) tasks, respectively. We model HI- and LO-security tasks based on the observation that some tasks are not essential to maintain safe or secure operation of the system, especially when the system may be under attack. This is depicted in Table I. For example, in an automotive system, infotainment services are not mission-essential functions, and should neither interfere with high security- or temporal-criticality tasks. Some tasks are also high criticality with respect to both security and temporal criticality, as they support mission-critical functionality. However, there are some tasks that could be critical to the security of the system, but be less critical to the temporal correctness of the system. For example, intrusion detection software or key management for encrypted communication may be critical to the security of the system, even if their timing is not mission critical. Alternatively, some sensor readings may support optional or non-mission-critical functionalities, which could be disabled in the presence of a security threat. However, in order to maintain consistent state, their processing is timing critical.

Note that LO-security tasks may themselves contain vulnerabilities. When the system is under attack, minimizing the attack surface is a valuable defense in and of itself. Given this motivation and model, we define the following terms:

Definition 1. ***(Victim Task and Targeted Task)*** *Any task $\tau_v \in \tau'$ is a victim task when it is attacked during runtime. As control-flow-hijacking attacks leverage one or more vulnerabilities within a single process only, we assume that an attack will target a single task. We assume the attack is detected by the process crashing as a result of a defensive mechanism such as CFI [1], [8], [24], as described in our threat model. Consequently, the attack is detected at or before the task completes its execution budget. We further denote a* HI*-security victim task $\tau_v \in \tau_\varsigma$ as a* Targeted Task*, τ_t, with an execution budget of C_t.*

Definition 2. ***(System Modes)*** *The system will begin its execution under* **normal mode***, during which no attack to security task is detected. During runtime, once a victim task is identified, the system will immediately switch into* **recovery mode***. Proper* actions *(see below) will be taken during recovery mode to prevent the system from further exploitation.*

When transitioning to the recovery mode, additional actions may be taken to facilitate recovery, for example, additional monitoring or validation of the system, forensic analysis, communication with human operators, *etc.* We model this additional workload as a *recovery task*. This task is in addition to the regular HI- and LO-security tasks as defined below:

TABLE II: Workload considered in Example 1.

Task ID	C_i	T_i	ς_i
τ_1	1	3	0
τ_2	2	9	1
τ_3	5	25	1
τ_R	1.5	15	–

Definition 3. ***(Recovery Task)*** *The recovery task $\tau_R = \{C_R, T_R\}$ is a task that is activated/released upon detection of an attack (which only leads to execution failure) during runtime, where C_R is its execution budget and T_R is the period. The release time of the recovery task, r_R, is equal to the system mode switch instant. Note that each* HI*-security task could have an individual recovery task. However, from the analytical perspective, taking $C_R = \max\{C_R^i\}$ covers the worst-case where C_R^i is the WCET of recovery task corresponding to i^{th}* HI*-security task.*

The whole SR^3 system workload contains the HI- and LO-security tasks, as well as the recovery task, *i.e.*, $\tau = \{\tau', \tau_R\}$.

Correctness Criteria. Given the SR^3 system, which contains a set of HI- and LO-security tasks and the recovery task, a *correct* scheduler must

1) guarantee that all HI- and LO-security tasks receive enough execution and meet their deadlines during normal mode;
2) ensure that all HI-security tasks (that are not experiencing any failure, *i.e.*, except the Targeted task) continue to receive normal execution budget and meet their deadlines during recovery mode;
3) ensure that if the victim task is a Targeted task (the failing HI-security[2] task being attacked), then the victim task will receive another full re-execution budget (of its original WCET, C_i) beyond the mode switch point and meets its original deadline;
4) provide the recovery task with enough execution budget before its deadline during recovery mode;

Our objective is to identify a correct online scheduling mechanism and derive an offline schedulability test. Note that once there is a detected attack (and thus a mode switch), guarantees to service of LO-security tasks are no longer required, and this workload is dropped to minimize the attack surface. We assume that the malicious input can be placed in a blocklist, and that either a known-safe or sanitized input is used by the re-executed job. This assumption is consistent with prior work [17].

Unfortunately, the standard uniprocessor scheduling algorithms (*e.g.*, Earliest Deadline First (EDF)) cannot correctly schedule the task set. An illustrative example is given below:

[2]When the victim task is a LO-security one, a mode switch is triggered immediately, while no re-execution budget will be allocated, as no guarantees are provided to LO-security tasks in recovery mode.

Example 1. *Consider a task set* $\tau = \{\tau_1, \tau_2, \tau_3, \tau_R\}$ *with parameters presented in Table II. This regular sporadic task set is schedulable on a uniprocessor system under the earliest deadline first (EDF) scheduler as the utilization* ($\sum \frac{C_i}{T_i}$) *of the task set is* 0.855 *(including the utilization of recovery task).*

Now let us map the SR^3 *system workload to a sporadic task model for EDF scheduling by doubling the execution of* HI*-security tasks,* ($C_2' = 4, C_3' = 10$) *and keep the recovery task always active. After mapping the task set to a sporadic task model for EDF scheduling, the utilization of the mapped task set becomes* 1.277*. Therefore, the mapped task set with security awareness is not schedulable by EDF. We will later see that the task set is schedulable under our proposed scheduling algorithm.*

III. Scheduling Algorithm

In this section, we present our proposed scheduling algorithms for the SR^3 system workload. We need to re-execute the targeted task after an adversarial attack following the task model. As demonstrated by Example 1, directly employing EDF scheduler may lead to a *too narrow* scheduling window for HI-security tasks upon attack and thus lead to deadline misses. Therefore, we proposed to adopt the concept of *virtual deadline*, such that the HI-security tasks receive proper 'promotion' under the normal mode.

Let us start with a general overview of our proposed algorithm. At any instant in normal mode, we aim to promote the execution of jobs of HI-security tasks over LO-security tasks, maintaining the deadline constraints of all tasks. To do so, we compute a virtual deadline $D_i^v = x \cdot D_i$ for each HI-security task such that the virtual deadline is less than or equal to the original deadline of the tasks (*i.e.*, no task exceeds the original deadline). After computing a suitable virtual deadline for each HI-security task, HI-security tasks are scheduled using their virtual deadline and LO-security tasks with their original deadline following the EDF algorithm. The window between the virtual and actual deadlines for each HI-security job is 'reserved' for the HI-security task's potential re-execution upon attack/mode switch. Further, in recovery mode, all LO-security tasks are dropped immediately at system mode-switch instant. Then, in recovery mode, all HI-security tasks and the recovery task are scheduled following the EDF algorithm using the original deadlines.

We present a way of determining the virtual deadline of the HI-security tasks and so the schedulability test of the scheduling algorithm. In Subsection III-A, we present the utilization-based schedulability analysis of the proposed algorithm (we denote it as sEDF-VD to distinguish it from EDF-VD [3]).

A. sEDF-VD

Given a sporadic implicit-deadline task set τ, one needs to perform a schedulability test for the task system prior to the runtime to determine whether the task system is schedulable or not. If the task system is schedulable, the sEDF-VD finds a virtual deadline $D_i^v = x \cdot D_i$ for all HI-security tasks via a common 'shrinking factor' $x \in (0, 1]$. In Algorithm 1, we present an offline schedulability test for the task set. The algorithm returns a common shrinking factor x for each HI-security task if the task set is schedulable, or FAILURE if the task set is not schedulable under the correctness criteria presented in Section II. The initial shrinking factor x in Line 1, Algorithm 1 comes from Lemma 1 and the conditional statement in Line 3, Algorithm 1 from Lemma 2. Both Lemmas are discussed in the following.

Algorithm 1: sEDF-VD based Schedulability Test (Virtual Deadline Setting)

Input: A SR^3 system workload $\tau = \{\tau_\varsigma, \tau_\natural, \tau_R\}$

1 $x \leftarrow \frac{U_\varsigma}{1-U_\natural}$; // common shrinking factor for all $\tau_i \in \tau_\varsigma$

2 **for** $\forall \tau_t \in \tau_\varsigma$ **do**

3 **if** $xU_\natural + U_\varsigma + u_t + u_R > 1$ **then**

4 return FAILURE; // no x that satisfy Theorem 1

5 **end**

6 **end**

7 return x;

We need to determine a feasible range of x and demonstrate its correctness. First we introduce additional notation.

Utilization parameters. The *utilization* of an implicit-deadline sporadic task is the ratio of its WCET (C_i) to the period (T_i). The utilization of the task system is the summation of all individual tasks in the set.

- $u_i = \frac{C_i}{T_i}$ is the utilization of task τ_i.
- $U_\natural = \sum_{\tau_i \in \tau_\natural} u_i$ is the utilization of LO-security task set.
- $U_\varsigma = \sum_{\tau_i \in \tau_\varsigma} u_i$ is the utilization of HI-security task set.

We will now derive the schedulability constraints of the SR^3 system workload considering the presence of a targeted task ($\tau_t \in \tau_\varsigma$) instead of any victim task through Lemma 1 and 2.

Lemma 1. *In* **normal mode**, *all jobs of* LO*-security tasks meet their actual deadline and all jobs of* HI*-security tasks meet their virtual deadline under sEDF-VD for the following sufficient inequality condition,*

$$x \geq \frac{U_\varsigma}{1 - U_\natural} \tag{1}$$

Proof Sketch. Let us consider a fluid schedule [14] (a conceptual scheduling scheme where each task gets a uniform execution rate over the scheduling period and the scheduler is schedulable if the total execution rate of all tasks is less than or equal to one), where each task in the task set is continuously assigned an execution rate of u_i for each LO-security task. Now, for HI-security tasks, we use virtual deadline deduced by multiplying the actual deadline by $x(\leq 1)$. Therefore, the fluid scheduler will assign a continuous execution rate of $\frac{u_i}{x}$ for each HI-security task. So, using the utilization bound of EDF [16], the task set will be schedulable if,

$$\sum_{\tau_i \in \tau_\natural} u_i + \sum_{\tau_i \in \tau_\varsigma} \frac{u_i}{x} = U_\natural + \frac{U_\varsigma}{x} \leq 1$$

So, the fluid schedule is feasible as the total utilization is less or equal to one. As the EDF in preemptive uniprocessor is optimal, the fluid schedule is also feasible by EDF.

Lemma 2. *In* **recovery mode**, *all* HI*-security tasks meet their actual deadlines, and the recovery task meets its deadline under EDF, if*

$$xU_{\setminus} + U_{\varsigma} + u_t + u_R \leq 1 \tag{2}$$

where u_t and u_R is the utilization of targeted task ($\tau_t \in \tau_{\varsigma}$) and recovery task, respectively.

Proof Sketch. Let us consider contrapositive. Suppose a job misses its deadline. Being in recovery mode, all LO-security jobs have been dropped and cannot miss a deadline. Therefore, the job missing its deadline must be a HI-security task. Let us consider a minimal instance[3] of jobs released by the task set, I, on which one job missed the deadline. Without loss of generality, we consider the earliest release time of a job in I (the last idle instant) as zero (0), and the deadline missed instant of a job in I as t_d. Let t^* denote the mode switch instant triggered by the job of targeted task τ_t. Note, the mode switch instant must be no later than the job's virtual deadline, *i.e.*, $t^* \leq D_t^v$.

Note that all jobs in I must experience some execution in $[0, t_d)$ except the job missed deadline at t_d. Let us consider the earliest release time of the job J amongst those executed in $[t^*, t_d)$ is a, and its deadline d. We will calculate the total executions of jobs in set I for four mutually exclusive subsets—subset of jobs from LO-security tasks, other HI-security tasks[4], the targeted task, and the recovery task separately.

Subset-1. Any LO-security task $\tau_i \in \tau_{\setminus}$ in I has an execution w_i in the considered scheduling window,

$$w_i \leq (a + x(t_d - a))u_i \tag{3}$$

Subset-2. Any *other* HI*-security task* $\tau_i \in \tau_{\varsigma} \setminus \tau_t$ in I has an execution w_i in the considered scheduling window,

$$w_i \leq \frac{u_i}{x}a + (t_d - a)u_i \tag{4}$$

Subset-3: The *targeted task* $\tau_t \in \tau_{\varsigma}$ has an execution,

$$w_t \leq \frac{a}{x}u_t + (t_d - a)2u_t \tag{5}$$

Subset-4. The *recovery task* τ_R has an execution of,

$$w_g \leq (t_d - a)u_R \tag{6}$$

Now, total execution of the jobs in I would be greater than the scheduling window length, t_d to miss a deadline.

$$\left(\sum_{\tau_i \in \tau_{\setminus}} w_i\right) + \left(\sum_{\tau_i \in \tau_{\varsigma} \setminus \tau_t} w_i\right) + w_t + w_g > t_d$$

$$\Rightarrow xU_{\setminus} + U_{\varsigma} + u_t + u_R > 1; \quad \text{(simplified using eq. 3,4,5,6)}$$

Thus, the minimal job instances, I will be schedulable if,

$$xU_{\setminus} + U_{\varsigma} + u_t + u_R \leq 1$$

[3]By minimal instance, we mean any set of jobs from which reducing one job would make the jobs set schedulable.

[4]By 'other HI-security tasks', we refer all HI-security tasks but the targeted task ($\tau_{\varsigma} \setminus \tau_t$).

Hence *Lemma 2* follows.

Using the Lemma 1 and 2, we get following scheduling test for sEDF-VD:

Theorem 1. *A* SR^3 *system workload τ, where the victim task is a targeted task, can be successfully scheduled by sEDF-VD on an uniprocessor if the following (sufficient) conditions hold:*

$$(A) : x \geq \frac{U_{\varsigma}}{1 - U_{\setminus}}; \quad \textit{[from Lemma 1]}$$

$$(B) : x \leq \frac{1 - U_{\varsigma} - u_t - u_R}{U_{\setminus}}, \forall \tau_t \in \tau_{\varsigma}; \quad \textit{[from Lemma 2]}$$

Example 2. *Let revisit the task set given in Example 1. Using Theorem 1 (A), we get $x \geq 0.633$ and for Theorem 1 (B), $x \leq 0.766$ for the task set. Note that, in condtion (B) of Theorem 1, we need to find the lowest value of x which can be found using $\max\{u_t | \tau_t \in \tau_{\varsigma}\}$ in calculation of x. So there is an x that satisfy both of the conditions of Theorem 1. Therefore, the task set is schedulable for sEDF-VD.*

Note that if we map the SR^3 system workload of Table II to mixed-criticality model [23] doubling the execution time of all HI*-security tasks in recovery mode ($C_i^n = C_i, C_i^e = 2C_i, \forall \tau_i \in \tau_{\varsigma}$) and the recovery task, τ_R as ($C_R^n = 0, C_R^e = C_R$), then the lower limit of shrinking factor x by Theorem 1 of [3] is 0.6333 and the upper limit by Theorem 2 of [3] is 0.1666. Therefore, there is no x that satisfy the schedulability constraints of mixed systems by EDF-VD presented in [3] for this mapped task set.*

IV. Evaluation

In this section, we evaluate our proposed algorithm. We will first explain the baseline algorithms that we consider. Next, we will present the workload generation procedure used to generate random task sets. Finally, we present the simulation results and discuss observations.

Baselines. The first baseline algorithm that we consider is the earliest deadline first (EDF) algorithm [16]. The EDF algorithm is used to schedule a workload that follows the sporadic task model and therefore to utilize the EDF algorithm, we map our proposed SR^3 system to the standard sporadic task model by doubling the utilization of each HI-security task in the system to account for the worst-case. Additionally, we also include the recovery task in the task set. With our proposed task model mapped to the sporadic task model, we use the utilization-based schedulability test for EDF algorithm to determine the schedulability.

The second baseline algorithm that we considered is the earliest deadline first with virtual deadline (EDF-VD) algorithm. EDF-VD algorithm is a widely accepted scheduler for the Vestal's mixed-criticality task model. We map our SR^3 system to the mixed-critical task model by allocating twice the utilization in the recovery mode for the HI-security tasks. We also add the recovery task as a HI-criticality task to the system where the normal execution budget of the recovery task is assumed to be 0. With these modifications, we apply the

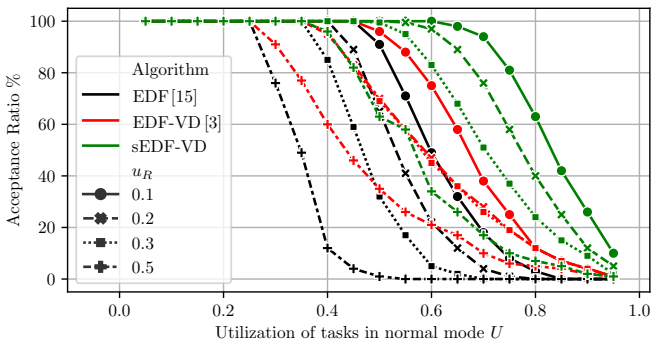

Fig. 1: Acceptance ratio of three different algorithms under multiple utilization settings

EDF-VD schedulability test [3] to determine the schedulability of the task set.

Workload generation. The SR^3 task set generation is controlled by the following parameters, where the default values are represented in bold.

- $n = \{5, \mathbf{10}, 15, 20\}$: Number of tasks in a task set
- $u_R = \{0.1, 0.2, \mathbf{0.3}, 0.5\}$: Utilization of the recovery task
- $U = U_\mathcal{L} + U_\varsigma = \{x/20 \mid 1 \leq x < 20\}$: Total utilization of the task set in normal mode
- $P = \{0.1, 0.2, \mathbf{0.5}, 1.0\}$: Probability of a task being HI-security task

The task set generation begins with a target value for normal mode utilization given by U. Using the UUniFast algorithm [5], we derive the set of task utilizations in normal mode. The recovery task utilization in recovery mode is given by the u_R parameter. For each setting, we generate 1000 task sets and present the results below.

Figure 1 reports the variation in acceptance ratios for varying system utilizations under different recovery task utilizations.

Observations. When applying EDF, it is seen in Figure 1 that, the acceptance ratio begins to drop as U increases beyond 0.5, irrespective of the recovery task utilization. This can be explained by the added pessimism to be considered by the EDF algorithm in the recovery mode. As the recovery task utilization increases, the performance further decreases. This behavior can be attributed to the added workload contributed by the increasing utilization of the recovery task.

When the EDF-VD algorithm is modified to schedule the proposed task model, the performance follows a similar trend as the EDF algorithm as shown in Figure 1. This pattern is consistent for all values of the recovery task. Similar to EDF, this observation can be attributed to the added pessimism in the higher criticality level due to the re-execution in the recovery mode. The pessimism arises from the need to cover the worst-case scenarios where a task re-execution can be triggered.

V. Conclusion

We have presented SR^3, a secure and resilient real-time attack recovery model, scheduler, and analysis. This model demonstrates that security criticality is an orthogonal dimension of criticality than has been studied in prior work on mixed-criticality scheduling. Our model is an example of a multi-mode mixed-criticality system, in which there are two modes, normal and recovery, and tasks are either high- or low-security criticality. Additionally, to facilitate recovery from a security event, a recovery task executes during recovery mode.

To avoid pessimism when adapting existing MC analysis, we developed a uniprocessor scheduling algorithm with modified virtual deadline for each HI-security task, and provided utilization-based schedulability test. Finally, we experimentally show that SR^3 performs better than the existing uniprocessor scheduling schemes such as EDF and EDF-VD for mixed-criticality systems upon model transformation via simulation on synthetic workload.

References

[1] M. Abadi et al. Control-flow integrity. In *ACM Conference on Computer and Communications Security*, CCS, 2005.

[2] M. Antonakakis et al. Understanding the mirai botnet. In *USENIX Security '17*. USENIX Association, Aug. 2017.

[3] S. Baruah et al. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS '12*, 2012.

[4] N. B. Bellec et al. RT-DFI: Optimizing data-flow integrity for real-time systems. In *ECRTS '22*, 2022.

[5] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] A. Burns. Multi-model systems — an mcs by any other name. In *8th International Workshop on Mixed Criticality Systems*, 2020.

[7] N. Burow et al. Moving target defense considerations in real-time safety- and mission-critical systems. In *Proceedings of the 7th ACM Workshop on Moving Target Defense*, pages 81–89, 2020.

[8] Y. Du et al. Holistic Control-Flow protection on Real-Time embedded systems with kage. In *USENIX Security '22*, 2022.

[9] J. Fellmuth et al. Instruction caches in static WCET analysis of artificially diversified software. In *ECRTS '18*, 2018.

[10] M. Hasan et al. Exploring opportunistic execution for integrating security into legacy hard real-time systems. In *RTSS '16*. IEEE, 2016.

[11] M. Hasan et al. Contego: An adaptive framework for integrating security tasks in real-time systems. *ECRTS '17*, 2017.

[12] M. Hasan et al. A design-space exploration for allocating security tasks in multicore real-time systems. In *DATE '18*. IEEE, 2018.

[13] M. Hasan et al. Period adaptation for continuous security monitoring in multicore real-time systems. In *DATE '20*. IEEE, 2020.

[14] P. Holman and J. H. Anderson. Adapting pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, 2005.

[15] G. Klein et al. seL4: Formal verification of an OS kernel. In *SOSP*, 2009.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[17] J. S. Mertoguno et al. A physics-based strategy for cyber resilience of cps. In *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2019*, 2019.

[18] S. Nagarakatte et al. SoftBound: Highly compatible and complete spatial memory safety for C. PLDI, 2009.

[19] C. Project. Memory safety, 2020.

[20] G. Serra et al. PAC-PL: Enabling control-flow integrity with pointer authentication in FPGA SoC platforms. In *RTAS '22*, 2022.

[21] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *CCS '07*, 2007.

[22] G. Thomas. A proactive approach to more secure code, 2019.

[23] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[24] R. J. Walls et al. Control-flow integrity for real-time embedded systems. In *ECRTS '19*, 2019.

[25] J. W. Wang et al. RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone. In *IEEE S&P*, 2022.

[26] B. C. Ward et al. The leakage-resilience dilemma. In *ESORICS 2019*, page 87–106.

[27] J. Zhou et al. Silhouette: Efficient protected shadow stacks for embedded systems. In *USENIX Security '20*, 2020.