Beyond Just Safety: Delay-aware Security Monitoring for Real-Time Control Systems*

MONOWAR HASAN, Wichita State University, USA
SIBIN MOHAN, Oregon State University, USA
RAKESH B. BOBBA, Oregon State University, USA
RODOLFO PELLIZZONI, University of Waterloo, Canada

Modern embedded real-time systems (RTS) are increasingly facing security threats than the past. A simplistic straightforward integration of security mechanisms might not be able to guarantee the *safety* and predictability of such systems. In this paper, we focus on integrating security mechanisms into RTS (especially *legacy* RTS). We introduce *Contego-C*, an analytical model to integrate security tasks into RTS that will allow system designers to improve the security posture without affecting temporal and control constraints of the existing real-time control tasks. We also define a *metric* (named tightness of periodic monitoring) to measure the effectiveness of such integration. We demonstrate our ideas using a proof-of-concept implementation on an ARM-based rover platform and show that Contego-C can improve security without degrading control performance.

CCS Concepts: • **Computer systems organization** → Real-time systems.

ACM Reference Format:
Monowar Hasan, Sibin Mohan, Rakesh B. Bobba, and Rodolfo Pellizzoni. 2022. Beyond Just Safety: Delay-aware Security Monitoring for Real-Time Control Systems. *ACM Transactions on Cyber-Physical Systems* 1, 1, Article 1 (March 2022), 25 pages.

1 INTRODUCTION

Embedded real-time systems (RTS) are pervasive and are found in everyday use, *e.g.,* automobiles, industrial and process control systems as well as in critical infrastructures (such as electrical grids, oil and gas infrastructures). RTS are also essential part of avionics and used in manned and unmanned aerial vehicles such as airplanes, drones, spacecraft. Given their application in safety

*This research is an extended version of work that was previously published in RTSS 2016 [1] and was conducted when M. Hasan and S. Mohan were affiliated with University of Illinois at Urbana-Champaign (UIUC). The major modifications/differences are: *(a)* we elaborate more on the system model and motivations for proposed mechanisms in Sections 1 and 2; *(b)* we introduce the notion of "control delay" and consider the performance of physical control system – this allows us to model the tasks with arbitrary deadlines (compared to prior work that was designed for implicit deadline systems only); *(c)* we generalize the framework (Section 2) to allow designers to execute security mechanisms within a predefined priority range (as opposed to executing at the lowest-priority as was proposed before [1]); *(d)* in addition to synthetic experiments, we include a proof-of-concept implementation on an ARM-based rover platform and evaluate the performance (Section 4.1); *(e)* because of the difference in semantics between the earlier work [1] and this one, we redesign the evaluation section (Section 4) with different parameters (*e.g.,* new rover platform and security tasks – refer to Table 5) specific to this work; *(f)* a new set of experiments (Section 4.2) compares the performance of the two approaches–the original one [1] and the one presented in this paper; *(g)* a new section (Section 6) presenting open research issues and possible extensions for our framework; and *(h)* other editorial changes to most sections, especially the abstract, introduction, related work and conclusion.

XXXX-XXXX/2022/3-ART1 $15.00
https://doi.org/

Table 1. Example of Security Tasks

Security Task	Approach/Tools
File-system checking	Tripwire [16], AIDE [17]
Network packet monitoring	Bro [18], Snort [19]
Hardware event monitoring	Statistical analysis based checks [20] using performance monitors (*e.g.,* perf [21], `OProfile` [22])
Application specific checking	Behavior-based detection (see the related work [13, 23–25])

critical domains, successful attacks or failures in RTS can have catastrophic consequences for the environment and/or to the human safety [2, 3].

Attack demonstrations by researchers on automobiles [3, 4] and medical devices [5] have shown that systems composed of RTS might be vulnerable to cyber-attacks. A number of high-profile attacks on real systems (*e.g.,* Stuxnet [6], BlackEnergy [7]) have shown that the threat is real. Traditional safety and fault-tolerance mechanisms used in RTS were designed to counter random or accidental faults and failures and cannot deal with intentional cyber attacks orchestrated by an intelligent and capable adversary. Further, the drive towards *(i)* use of standardized protocols and common-off-the-shelf (COTS) components for interoperability reduced infrastructure and maintenance costs, and *(ii)* smart and connected systems (*e.g.,* smart and connected communities, smart grids, smart or cyber manufacturing, smart transportation) is reducing any protection against cyber attacks that the use of proprietary components and being air-gapped (*i.e.,* unconnected to external systems) might have provided.

Recognizing this emerging threat and urgent need, there has been a lot of focus on securing RTS in the recent past including integrating communication confidentiality [8, 9], communication integrity [10, 11] and monitoring and detection mechanisms [1, 12–14]. When integrating any security mechanisms into RTS, the designers need to ensure that they do not perturb or impact the real-time functions in any significant way while at the same time provide the necessary level of security. Integrating security is especially challenging for those *legacy* RTS[1] where the schedule of the existing tasks cannot easily be changed.

In this paper, we aim to improve the security posture of RTS through integration of "security tasks" (*e.g.,* tasks that are specific for intrusion monitoring and detection tasks purposes) into an existing fixed-priority system while ensuring that the existing real-time/control tasks are not affected by such integration. Security tasks could include protection, detection or response mechanisms, depending on the system requirements – for instance, a sensor correlation task (to detect sensor manipulation) or an anomaly detection task (that checks possible intrusions) [15]. In Table 1 we present some examples of security tasks that can be integrated into legacy systems (again, this is by no stretch meant to be an exhaustive list). In our experiments, we considered intrusion detection as a monitoring mechanism and used Tripwire [16] (a data integrity checking tool) to demonstrate the feasibility of our approach – the ideas presented here though apply more broadly to other security mechanisms.

We proposed a design-time model (named *Contego-C*) that enables system designers to carefully *trade-off the effectiveness of security tasks with the control performance* of some pre-selected low priority real-time tasks without impacting the schedule and performance of high priority real-time

[1] A legacy RTS is one where modification or perturbation of existing real-time tasks' parameters (such as run-times, period and task execution order) is not always feasible.

tasks. For example, consider the integration of an intrusion detection system (IDS) in an existing RTS (for instance Tripwire or AIDE from Table 1) that checks integrity of filesystems. For functional correctness, the IDS task needs to execute at least once within a certain time period. If such a task is scheduled less frequently or interrupted often before it can complete checking the entire system (say by other, higher priority, real-time or control tasks), then an adversary could use that opportunity to intrude into the system and modify sensitive file contents before the next invocation of the detection task. In contrast, if the IDS task is executed more frequently, it may interfere the operation of other low priority tasks. Our analysis engine takes the real-time task parameters and periodicity requirements of the security tasks and then find the suitable periods and priority for the security tasks without violating timing requirements (refer to Section 3 for a formal model). This is different than criticality-monotonic priority scheduling [26] in mixed-criticality systems [27] where task period and priority orders are already defined.

Contego-C generalizes our previous work [1] on *opportunistic execution* of security tasks in two important ways. *First*, unlike prior work where security tasks can only run during slack times (*i.e.,* at the lowest priority relative to all the real-time tasks), Contego-C allows security tasks to run at a priority higher than some pre-selected low priority real-time/control tasks. This allows for improved effectiveness (performance) of security tasks while not degrading the control performance of the real-time tasks below the required thresholds. *Second*, real-time tasks do not have any specific assumption on deadlines (*i.e.,* implicit deadlines as was the case in earlier work [1]). When used with a legacy RTS with implicit deadlines and when the set of low priority tasks whose schedule is allowed to be perturbed is set to empty, Contego-C falls back to opportunistic execution. Otherwise, Contego-C provides the opportunity to improve the performance of security tasks without degrading the system's real-time/control performance – thus subsumes our earlier work.

The main contributions of this work can be summarized as follows.

- A design-time tool: Contego-C, for fixed-priority RTS that allows security tasks to execute in conjunction with real-time control tasks without degrading control performance. Contego-C is suitable for legacy systems where designers have less flexibility to change system parameters, perhaps due to control requirements.
- A mathematical model (Sections 3.1 and 3.2) and an iterative algorithm (Section 3.3) that allows security tasks to execute with a frequency closer to the desired one without violating real-time/control requirements of the other real-time tasks.
- A proof-of-concept implementation on an ARM-based surveillance rover that demonstrates the trade-off between security and real-time requirements (Section 4.1).

We also evaluate Contego-C with synthetic workloads for schedulability and security (Section 4.2) and show the effectiveness of our security integration framework.

2 SYSTEM AND SECURITY MODEL

In the following, we introduce our system model (Section 2.1), clarify the threat model (Section 2.2), give an overview of the problem that we address in this paper (Section 2.3) and present security task parameters (Section 2.4). Key mathematical notations used in the paper are listed in Table 2.

2.1 Real-Time Task Model and Control Costs

2.1.1 Real-Time Tasks. Let us consider a single core platform where we schedule a set $\Gamma_R = \{\tau_1, \tau_2, \cdots, \tau_{N_R}\}$ of N_R independent sporadic real-time/control tasks.[2] Each task $\tau_r \in \Gamma_R$ is characterized by the following parameters: *(a)* the real-time parameters – describe how the task interacts

[2]We use the terms *real-time task* and *control task* interchangeably throughout the paper.

Table 2. Mathematical Notations

Notation	Interpretation
Γ_R, Γ_S	Set of real-time and security tasks, respectively
N_R, N_S	Number of real-time and security tasks, respectively
C_i, T_i	Worst-case execution time, and period of (real-time and security) task τ_i, respectively
x_r, u_r, v_r, y_r	State, input, disturbance and output of the plant for task τ_r, respectively
J_r, J_r^{TH}	Linearized control cost and cost threshold of the real-time task τ_r, respectively
Δ_r	Control delay of the real-time task τ_r
ω_s	Weighting factor for security task τ_s
$[l_S, N_R]$	Allowable priority-level for security tasks
$hp_R(\tau_i)$, $hp_S(\tau_i)$	Set of real-time and security tasks that has higher and lower priority than τ_i, respectively
$hp_R^l(\tau_s)$	Set of real-time tasks that has higher priority than security task τ_s (for a given priority-level l)
$lp_R^l(\Gamma_S)$	Set of real-time tasks that has lower priority than security tasks (for a given priority-level l)
T_s^{des}, T_s^{max}	Desired and maximum allowable period of the security task τ_s, respectively
η_s	Tightness of the periodic monitoring for security task τ_s
I_s	Upper bound of the interference experienced by τ_s
ξ	Normalized difference between achievable and desired periods
$W_s(J^{TH})$	Weighted schedulability metric for a given cost threshold J^{TH}

with the scheduler and the other tasks; *(b)* the control parameters – describe the plant, the controller and the *quality of the control* (*e.g.,* control cost). We represent each real-time task τ_r by the tuple $(C_r, T_r, J_r, J_r^{TH})$ where C_r is the worst-case execution time (WCET), T_r is the minimum separation (*e.g.,* period) between two successive invocations. The variables J_r and J_r^{TH} represent the control cost and cost threshold (for acceptable control performance), respectively. A preemptive fixed-priority scheduler manages the tasks based on task priority. We use the variable $hp_R(\tau_r)$ to represent the set of real-time tasks that have a priority higher than τ_r. We assume that real-time task priorities are distinct and the periods and the priorities are independent parameters (*e.g.,* specified by the system designer). We further assume that context switch overhead and cache-related preemption delays are either *(a)* negligible compared to WCET of the task and/or *(b)* included in the WCET measurements and independent of underlying scheduling policy [28, 29].

2.1.2 Control System Preliminaries. We consider a linear-quadratic Gaussian (LQG) framework [30] to model the control system where each plant P_r is described as follows [31–34]:

$$\begin{aligned} \frac{\mathrm{d}x_r(t)}{\mathrm{d}t} &= A_r x_r(t) + B_r u_r(t) + v_r(t) \\ y_r(t_k) &= C_r x_r(t_k) + e_r(t_k) \end{aligned} \tag{1}$$

In Eq. (1), x_r and u_r are the plant state and controlled input, respectively and v_r is the plant disturbance (*e.g.,* continuous-time white-noise process). The output y_r is measured at t_k and used to produce the control signal u_r. The measurement noise e_r is modeled as a discrete-time Gaussian white-noise process. The control performance is measured by the following quadratic cost function [32]:

$$J_r = \lim_{t\to\infty} \frac{1}{t} \mathbb{E}\left\{ \int_0^t (y_r^2(\sigma) + \rho_r u_r^2(t))\, \mathrm{d}\sigma \right\} \tag{2}$$

where $\mathbb{E}$ denotes the expectation operation and ρ_r is a designer-provided weighting factor for the magnitude of the plant states and the control signals. The variable J_r can be used as a measure of performance loss since the quality of a controller is degraded (*i.e.,* cost is increased) if the delay (task response time) is different from what was assumed during the control-law synthesis. For a given period T_r and control delay Δ_r (*e.g.,* due to execution of the task and preemption by other high-priority tasks), the control cost in Eq. (2) can be approximated by the following linear function [32, 34]:

$$J_r = \alpha_r T_r + \beta_r \Delta_r \tag{3}$$

where the parameters α_r and β_r are the weighting factors used for linearization and can be obtained from the physical properties of the plant.[3]

2.1.3 System Schedulability Conditions. We assume that the real-time tasks are *schedulable* (*e.g.,* control cost is within an acceptable threshold) by a fixed-priority scheduling policy and that the following condition holds[4]:

$$\alpha_r T_r + \beta \Delta_r \leq J_r^{TH} \tag{4}$$

where the control delay Δ_r can be obtained by traditional response time analysis [29, 35] as follows.

Let us denote q as an index to each job within the *busy-window*.[5] The completion time of each job of the task τ_r can be computed as follows:

$$\delta_r(q) = (q+1)C_r + \sum_{\tau_h \in hp_R(\tau_r)} \left\lceil \frac{\delta_r(q)}{T_h} \right\rceil C_h. \tag{5}$$

where the summation term represents the interference from other high-priority tasks. The above equation can be solved by recurrence, *e.g.,*

$$\delta_r(q)^k = (q+1)C_r + \sum_{\tau_h \in hp_R(\tau_r)} \left\lceil \frac{\delta_r(q)^{k-1}}{T_h} \right\rceil C_h$$

with $\delta_r(q)^0 = (q+1)C_r$ and $\delta_r(0)^0 = C_r$. For a given q, the recurrence will terminate if $\delta_r(q)^k = \delta_r(q)^{k-1}$ for some k. The control delay (*e.g.,* response time) of each job q is calculated by:

$$\Delta_r(q) = \delta_r(q) - qT_r. \tag{6}$$

[3]Table 7 presents these parameters for three automotive control systems.

[4]Note that when $\alpha_r = 0$ and $\beta_r = 1$, Eq. (4) can be mapped to the traditional real-time schedulability condition used in real-time literature (*e.g.,* $R_r \leq D_r$ where R_r and D_r are the response-time and deadline of the task τ_r, respectively).

[5]A busy-window [35] of τ_r is the interval $[t_0, t]$ within which jobs with priority higher than τ_r are processed throughout $[t_0, t]$ but no jobs with priority higher than τ_r are processed in $t_0 - \epsilon, t_0)$ or $(t, t + \epsilon)$ for a sufficiently small ϵ.

Therefore the worst-case control delay is given by:

$$\Delta_r = \max_{\forall q=\{0,1,\cdots\}} \{\Delta_r(q)\}. \tag{7}$$

When the system is not overloaded (*e.g.,* $\sum_{\tau_r \in \Gamma_R} \frac{C_r}{T_r} \leq 1$), the iteration of increasing values of q will stop when $\delta_r(q) \leq (q+1)T_r$.

2.2 Threat Model

We assume that an adversary may destabilize the system. For example, an attacker could compromise the file system (resulting in corrupted information/system log), change the of control/actuation commands or infer side-channel information (*e.g.,* user tasks, cache information, thermal profiles, *etc.*) to launch further attacks (say denial of service). While there exist mechanisms (such as Simplex [36, 37]) that guarantee (hardware/software) fault tolerance, we consider the cases where an attacker intentionally induces faults (*i.e.,* adversarial artifacts) that may jeopardize the safety of the system (*e.g.,* results in miss deadlines). Our focus is on threats that can be dealt with by integrating additional security tasks into the host. The addition of such tasks may necessitate changing the schedule or increasing the WCET of real-time tasks as was the case in earlier work [8, 9, 38–40]. In this research, we consider situations where additional security tasks (see Table 1 for related examples) are only allowed to have minimal or no impact on the schedule of existing real-time tasks and are not allowed to modify real-time/control parameters. While we use specific intrusion detection mechanisms (*e.g.,* Tripwire) to demonstrate our approach, Contego-C is agnostic to the specific monitoring mechanism. The design of Contego-C and the design of the specific security tasks are orthogonal problems. Since we aim to maximize the frequency of execution of security tasks, security mechanisms whose performance improves with the frequency of execution (*e.g.,* intrusion monitoring and detection tasks or logging/tracing mechanisms) benefit from our model.

2.3 Considerations for Integrating Security Mechanisms

We consider incorporating security mechanisms by implementing them as separate *sporadic tasks*. In order to provide the best protection, these security tasks may need to be executed quite often. If the interval between consecutive monitoring events is too large, the adversary may harm the system (and remain undetected) between two invocations of the security task. On the other hand, if the security tasks are executed very frequently then it may impact the schedulability of the real-time tasks – herein lies an important trade-off between monitoring frequency and schedulability of real-time/security tasks. Specifically, this brings up the challenge of determining the *right periods* (*viz.,* minimum inter-execution time) for the security tasks [41]. For instance, some critical security tasks may be required to execute more frequently than others. However, if the period is too short (*e.g.,* the security task repeats too often) then it will use too much of the processor time and eventually lower the overall system utilization (and consequently the performance of the underlying control system). As a result, the security mechanism itself might prove to be a hindrance to the system and reduce the overall functionality or worse, safety. In contrast, if the period is too long, the security task may not always detect violations since attacks could be launched between two instances of the security task. Besides, if the security tasks execute with lower priority, they suffer more interference (*i.e.,* preemption from high-priority control tasks) and the longer detection time (due to poor response time) will make the security mechanisms less effective.

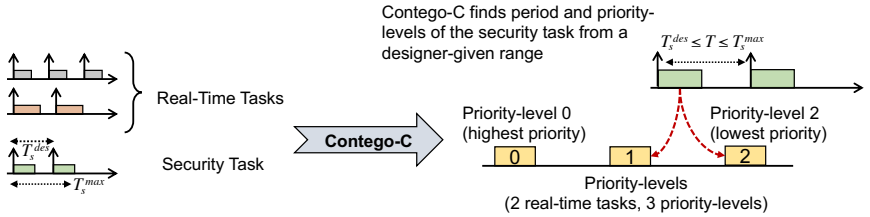

Fig. 1. High-level overview of Contego-C. In this illustration, we consider that one security task is integrated into the system consisting of two real-time tasks where the security task can execute up to priority-level 1. Contego-C finds the appropriate period and priority-level for security task in a way that timing requirements of real-time tasks are not violated (refer to Section 3 for a formal description).

2.4 Security Tasks

In this research, we focus on legacy systems where designers may not have enough flexibility to modify system parameters to integrate security mechanisms. Hence we want to *ensure security without any modification of real-time/control task parameters and have minimal impact on the task schedule*. Figure 1 presents a high-level schematic of Contego-C. In this illustration, we integrate one security task and finds the suitable period and priority-level such that the timing constraints of two existing real-time tasks are not affected. As we mentioned earlier, we integrate security tasks as independent sporadic tasks. Let us consider additional N_S security tasks denoted by the set $\Gamma_S = \{\tau_1, \tau_2, \cdots, \tau_{N_S}\}$. We follow the sporadic security task model [1] and characterize each security task τ_s by the tuple $(C_s, T_s^{des}, T_s^{max}, \omega_s)$ where C_s is the WCET, T_s^{des} is the best period (minimum inter-arrival time) between successive releases (*i.e.,* $F_s^{des} = \frac{1}{T_s^{des}}$ is the desired frequency for τ_s for effective security monitoring and/or intrusion detection), T_i^{max} is the maximum period beyond which security monitoring will not be effective, and $\omega_s > 0$ is a weighting factor (where $\sum_{\tau_s \in \Gamma_S} \omega_s = 1$) represents the severity of the security tasks (more critical task could have higher weight). We assume that security tasks follow a designer specified fixed-priority order and have implicit deadlines (*e.g.,* they are required to complete execution before its period).

We assume that security tasks are allowed to execute with a priority higher than *certain* low-priority real-time tasks. Since the task priorities are distinct, there are N_R priority-levels for real-time tasks (indexed from 0 to $N_R - 1$ where level 0 is the highest priority). Among the N_R priority-levels, we assume that security tasks can execute up to priority-level l_S $(0 < l_S \le N_R)$, $l_S \in \mathbb{N}$. Notice that $l_S = N_R$ implies that the security tasks execute with the lowest priority and are allowed to run *only* during slack times when other real-time tasks are not running.

One fundamental problem in integrating security tasks is to determine *which* security tasks will be running *when* [41]. Although any period T_s within the range $T_s^{des} \le T_s \le T_s^{max}$ and priority-level $l \in [l_S, N_R]$ would be acceptable, the actual period T_s and priority-level l however, is not known a priori. Therefore our goal is to find *suitable periods as well as priority-levels for the security tasks* without violating the real-time constraints.

One may wonder why we cannot assign the desired period (*e.g.,* $T_s = T_s^{des}$) set the priority-level as $l = l_S$ so that the security tasks can always execute with the desired frequency F_s^{des} and experience less interference (*e.g.,* preemption) from real-time tasks. However, without careful schedulability analysis if we set $l = l_S$ (or arbitrarily from the range $[l_S, N_R]$) and $T_s = T_s^{des}$, $\forall \tau_s$ this may violate real-time constraints for the control tasks – thus the main safety requirements of the system will be threatened.

3 PERIOD AND PRIORITY SELECTION

Since the actual period as well as the priority-levels of the security tasks are unknown, these values need to be calculated; we must also ensure that they fall within acceptable ranges. Let T_s be the period of the security task $\tau_s \in \Gamma_S$ that needs to be determined. Since our goal is to run the security tasks with a period as close to the desired period (T_s^{des}) as possible, without impacting the real-time tasks (*i.e.,* control system performance), we use the following metric [1]:

$$\eta_s = \frac{T_s^{des}}{T_s}, \tag{8}$$

that denotes the *tightness* of the frequency of periodic monitoring for the security task τ_s. Thus $\eta = \sum_{\tau_s \in \Gamma_S} \omega_s \eta_s$ denotes the *cumulative tightness* of the achievable periodic monitoring. This monitoring frequency metric, for instance, provides one way to measure the trade-offs between security and schedulability – since this metric η will allow us to execute the security routines with a frequency closer to the desired one while respecting the temporal and control constraints of the real-time tasks.

3.1 The Formulation as an Optimization Problem

Recall from our earlier discussion that our goal is to ensure that security tasks can execute with a period close to what the designers' expect (*i.e.,* T_s^{des}). In doing so, we also need to ensure that *(a)* security tasks are schedulable (*i.e.,* the finish execution before their next periodic invocation), *(b)* the timing requirements of real-time tasks are satisfied and *(c)* the periods of the security tasks are within designer-specified bound (*i.e.,* $[T_s^{des}, T_s^{max}]$). We formulate this as an optimization problem that maximizes our tightness metric with respect to the three constraints listed above. In particular, for a given priority-level[6] $l \in [l_S, N_R]$, we can represent the period selection as a *constrained optimization problem* as we describe in the following.

3.1.1 Objective Function. The objective of the period selection is to minimize the perturbation (*e.g.,* maximize the tightness η_s) for all the security tasks. Mathematically the objective function can be defined as follows:

$$\max_{\mathbf{T}} \sum_{\tau_s \in \Gamma_S} \omega_s \frac{T_s^{des}}{T_s} \tag{9}$$

where the vector $\mathbf{T} = [T_1, T_2, \cdots, T_{N_S}]^\mathsf{T}$ represents the periods of the various security tasks that need to be determined.

3.1.2 Schedulability Constraints for Security Tasks. Since the security tasks are executed with a priority lower than some real-time tasks, they will suffer interference from real-time and other high-priority security tasks. For a given priority-level l, let us denote $hp_R^l(\tau_s) \subset \Gamma_R$ the set of real-time tasks that are with a priority higher than τ_s. Also, let $hp_S(\tau_s) \subset \Gamma_S$ denote the set of higher-priority security tasks than τ_s. The worst-case release pattern of τ_s occurs when τ_s and all high-priority (real-time and security) tasks are released simultaneously [42]. Using response time analysis [43] we can determine an upper bound on the interference experienced by τ_s as follows:

$$I_s = \sum_{\tau_r \in hp_R^l(\tau_s)} \left(\frac{T_s}{T_r} + 1 \right) C_r + \sum_{\tau_h \in hp_S(\tau_s)} \left(\frac{T_s}{T_h} + 1 \right) C_h \tag{10}$$

where the first and second terms represent the amount of interference from real-time and high-priority security tasks, respectively. In order to ensure that each security task τ_s will complete

[6]Selection of security tasks' priority-level is described in Section 3.3.

execution before its next invocation, the following constraint needs to be satisfied:

$$C_s + I_s \leq T_s, \ \forall \tau_s \in \Gamma_s. \tag{11}$$

Notice that for a given priority-level $l \in [l_S, N_R]$ the set of high-priority real-time tasks (*e.g.,* $hp_R(\tau_s)$) is fixed. Therefore T_s is the only variable in Eqs. (10) and (11).

3.1.3 Schedulability Constraints for Low-Priority Real-Time Tasks. As mentioned earlier, the security tasks can be executed at any priority within the range $[l_S, N_R]$. For a given priority-level $l \in [l_S, N_R]$, there are $N_R - l$ real-time tasks that have a lower priority than the security tasks and will suffer interference from other high-priority real-time and security tasks. Note that control delay of the real-time tasks that are with a priority higher than security tasks will not be affected and hence schedulability conditions of those tasks are ensured by assumption. However, we need to ensure that the control cost is within the allowable threshold (*e.g.,* the condition in Eq. (4) is satisfied) for the real-time tasks that are executing with a priority lower than the security tasks. For a given priority-level l, let us denote $lp_R^l(\Gamma_S)$ as the set of real-time tasks that have a lower priority than the security tasks. Hence we define the following constraints to ensure the *schedulability of the low-priority real-time tasks*[7]:

$$J_r = \alpha_r T_r + \beta_r \hat{\Delta}_r \leq J_r^{TH}, \ \forall \tau_r \in lp_R^l(\Gamma_S). \tag{12}$$

In the above constraints $\hat{\Delta}_r$ is the control delay and defined as:

$$\hat{\Delta}_r = \max_{\forall q = \{0,1,\cdots\}} \{\delta_r(q) - qT_r\} \tag{13}$$

where

$$\delta_r(q) = (q+1)C_r + \sum_{\tau_h \in hp_R(\tau_r)} \left\lceil \frac{\delta_r(q)}{T_h} \right\rceil C_h + \mathbb{I}(\tau_r) \times \sum_{\tau_s \in \Gamma_S} \left\lceil \frac{\delta_r(q)}{T_s} \right\rceil C_s \tag{14}$$

and the binary variable $\mathbb{I}(\tau_r) = 1$ if $\tau_r \in lp_R^l(\Gamma_S)$. The last summation term in Eq. (14) represents the additional interference introduced by the security tasks.

The constraints in Eq. (12), in the current form, require us to solve a recurrence with unknown period T_s for all the security tasks $\tau_s \in \Gamma_S$. Such constraints make the optimization problem intractable to solve. We address this issue by considering an upper bound of control delay and rewrite the constraints in Eq. (12) as follows:

$$J_r = \alpha_r T_r + \beta_r \Delta_r \leq J_r^{TH}, \ \forall \tau_r \in lp_R^l(\Gamma_S). \tag{15}$$

where the control delay Δ_r is given by:

$$\Delta_r = \tilde{q}_r C_r + \sum_{\tau_h \in hp_R(\tau_r)} \left\lceil \frac{\tilde{\Delta}_r}{T_h} \right\rceil C_h + \mathbb{I}(\tau_r) \times \sum_{\tau_s \in \Gamma_S} \left(\frac{\tilde{\Delta}_r}{T_s} + 1 \right) C_s. \tag{16}$$

In the above equation $\tilde{q}_r = \left\lceil \frac{\tilde{\Delta}_r}{T_r} \right\rceil$ and $\tilde{\Delta}_r$ denotes an upper bound of response time (*e.g.,* the maximum size of the busy-window)[8]. We can calculate this upper bound $\tilde{\Delta}_r$ by using Eq. (13) and by setting $T_s = T_s^{des}, \ \forall \tau_s \in \Gamma_S$ in Eq. (14) – however this may cause response time unbounded for some T_s (*e.g.,* if the total system utilization greater than unity for $T_s = T_s^{des}$). Note that for any schedulable system the control cost is upper bounded by J_r^{TH} (see Eq. 15). We therefore calculate

[7]Note that if $l = N_R$ (*e.g.,* security tasks are running with the lowest priority), the constraints in Eq. (12) are no longer necessary.

[8]Note that if $\tilde{\Delta}_r \leq T_r$ (*e.g.,* constrained deadline systems), then $\tilde{q}_r = 1$ and Δ_r in Eq. (16) becomes an upper bound [43] of standard response time expression [42].

$\tilde{\Delta}_r$ by using the minimum of $\frac{J_r^{TH} - \alpha_r T_r}{\beta_r}$ (*e.g.,* by rearranging Eq. 15) and the expression in Eq. (13) (where we set $T_s = T_s^{des}$ in Eq. (14)). Note that this delay bound Δ_r in Eq. (16) could be pessimistic, especially for lower-priority real-time tasks, however as we show in Section 4 this allows us to integrate security tasks with reasonable performance while not degrading control performance beyond an acceptable limit.

3.1.4 Period Bound Constraints. In order to guarantee the restrictions on monitoring frequency (*e.g.,* the periods are within the designer specified bounds), the following inequality needs to be satisfied for all the security tasks:

$$T_s^{des} \leq T_s \leq T_s^{max} \quad \forall \tau_s \in \Gamma_S. \tag{17}$$

Remark. *For a feasible solution, the periods of the security tasks could be any value within the range* $[T_s^{des}, T_s^{max}]$, $\forall \tau_s \in \Gamma_S$ *that respect all the constraints. Therefore, the mathematical formulation of period selection with objective function in Eq. (9) and constraints in Eqs. (11), (15) and (17) is a constrained combinatorial optimization problem (e.g., for a given priority-level* l*, there are* N_S *variables and* $2N_S + (N_R - l)$ *constraints).*

3.2 Solution to the Period Selection Problem

We solve the period selection problem introduced in Section 3.1 by transforming it into a geometric program (GP) [44]. The key idea is to transform the objective function and the constraints into equivalent GP form that can be solved using existing techniques (*e.g.,* known algorithms such as the *interior-point* method [45, Ch. 11] or standard convex optimization solvers [46, 47]). We now first briefly introduce GP (Section 3.2.1) and then describe our approach to solve the period selection problem (Sections 3.2.2 and 3.3).

3.2.1 Preliminaries of Geometric Program. A constrained optimization problem can be solved by GP if the problem is formulated as follows [44]:

$$\begin{aligned} &\min_{\mathbf{Y}} f_0(\mathbf{y}), \\ &\text{Subject to: } f_i(\mathbf{y}) \leq 1, \quad i = 1, \cdots, z_p, \text{ and} \\ &g_i(\mathbf{y}) = 1, \quad i = 1, \cdots, z_m \end{aligned}$$

where $\mathbf{y} = [y_1, y_2, \cdots, y_z]^{\mathrm{T}}$ denotes the vector of z optimization variables. The functions $g_1(\mathbf{y}), \cdots, g_{z_m}(\mathbf{y})$ are *monomial* and $f_0(\mathbf{y}), f_1(\mathbf{y}), \cdots, f_{z_p}(\mathbf{y})$ are *posynomial* functions, respectively. A monomial function is expressed as

$$g_i(\mathbf{y}) = c_i \prod_{k=1}^{K_i} y_k^{a_k}, \tag{19}$$

where $c_i \in \mathbb{R}^+$ and $a_k \in \mathbb{R}$. Note that the coefficient c_i must be non-negative but the exponents a_l can be any real number including fractional and negative. A posynomial function (*i.e.,* the sum of the monomials) can be represented as

$$f_i(\mathbf{y}) = \sum_{k=1}^{K_i} c_k y_1^{a_{1k}} y_2^{a_{2k}} \cdots y_z^{a_{1k}}, \tag{20}$$

where $c_k \in \mathbb{R}^+$ and $a_{jk} \in \mathbb{R}$, $1 \leq k \leq K_i$. We can maximize a non-zero posynomial objective function by minimizing its inverse. In addition, we can express the constraint $f(\cdot) < g(\cdot)$ as $\frac{f(\cdot)}{g(\cdot)} \leq 1$.

Observation. *The period selection problem can be reformulated as a standard geometric program.*

3.2.2 Solving Period Section Problem as a GP. In order to represent the period selection problem as a GP, we rewrite the objective function in Eq. (9) as:

$$\min_{\mathbf{T}} \left(\omega_s T_s^{des}\right)^{-1} T_s. \tag{21}$$

The schedulability constraints for security tasks (Section 3.1.2) can then be expressed as:

$$(C_s + I_s) T_s^{-1} \leq 1 \tag{22}$$

where

$$I_s = \sum_{\tau_r \in hp_R^l(\tau_s)} \left(T_s + T_r\right) T_r^{-1} C_r + \sum_{\tau_h \in hp_S(\tau_s)} \left(T_s + T_h\right) T_h^{-1} C_h.$$

Similarly, we rewrite the schedulability constraints for real-time tasks that are with a priority lower than l (Section 3.1.3) as follows:

$$\frac{\alpha_r T_r + \beta_r \Delta_r}{J_r^{TH}} \leq 1, \ \forall \tau_r \in lp_R^l(\Gamma_S) \tag{23}$$

where $\Delta_r = \left\lceil \frac{\tilde{\Delta}_r}{T_r} \right\rceil C_r + \sum_{\tau_h \in hp_R(\tau_r)} \left\lceil \frac{\tilde{\Delta}_r}{T_h} \right\rceil C_h + \mathbb{I}(\tau_r) \times \sum_{\tau_s \in \Gamma_S} \left(\tilde{\Delta}_r + T_s\right) T_s^{-1} C_s$. Finally the period bound constraints (Section 3.1.4) can be represented as:

$$T_s^{des} T_s^{-1} \leq 1, \ \forall \tau_s \in \Gamma_s \tag{24}$$

$$(T_s^{max})^{-1} T_s \leq 1, \ \forall \tau_s \in \Gamma_s. \tag{25}$$

Using logarithmic transformations (*e.g.,* representing $\tilde{T}_s = \log T_s$ and hence $T_s = e^{\tilde{T}_s}$, and replacing inequality constraints of the form $f_i(\cdot) \leq 1$ with $\log f_i(\cdot) \leq 0$), we can convert the above formulation into a convex optimization problem (refer to [1, 44] for details). This transformed problem can be solved using standard algorithms[9] (such as *interior-point* method) in polynomial time [45, Ch. 11].

3.3 Algorithm

We develop an iterative scheme (Algorithm 1) to jointly obtain the periods and priority-level of the security tasks. The workflow of the algorithm is as follows.

We first solve the period selection problem (Section 3.2) for each of the allowable priority-levels (Line 2-9). If there exists a solution for any priority level $l' \in [l_S, m]$ (*e.g.,* the optimization problem is feasible with the given constraints), we store the periods in a candidate solution list (Line 6). If the candidate solution list is non-empty (*i.e.,* the problem is feasible for at least one priority-level), we then find the best priority-level (say l^*) from the candidate list that maximizes the cumulative tightness (Line 12) and return the tuple $\{l^*, \mathbf{T}(l^*)\}$, *i.e.,* priority-level and periods for security tasks (Line 14). If no candidate solutions are found (*e.g.,* the boolean flag Schedulable is false), the task-set is reported as unschedulable (Line 16) since it is not possible to integrate the given security tasks with desired requirements. This unschedulability result will provide hints to the engineers to modify the system parameters/requirements (*e.g.,* desired/maximum period of security tasks, number of security tasks, allowable priority-level) for integrating security mechanisms.

4 EVALUATION

We evaluate the performance of our security integration framework on two fronts: *(a)* a proof-of-concept implementation on an ARM-based surveillance rover – to demonstrate the viability of the proposed scheme in a practical cyber-physical system (Section 4.1) and *(b)* experiments with synthetically generated workloads – for a broader design-space exploration (Section 4.2).

[9]There also exist open-source software packages [46, 47] that can solve this GP problem efficiently.

Algorithm 1 Feasibility Checking and Parameter Selection

Input: Set of real-time and security tasks, Γ_R and Γ_S, respectively, and allowable priority ranges $[l_S, N_R]$
Output: The tuple $\{l^*, \mathbf{T}(l^*)\}$, *e.g.,* priority-level and periods of the security tasks if the task-set is schedulable; Unschedulable otherwise

```
Schedulable := false /* A boolean flag */
for each priority-level l' ∈ [l_S, N_R] do
    Solve the period selection problem (Section 3.2)
    if SolutionFound then
        /* store the periods T* (obtained by solving the optimization problem) for priority-level l' */
        T(l') := T*
        Schedulable := true
    end if
end for
/* Select periods and priority-level that provide maximum tightness */
if Schedulable then
    Find the priority-level l* from T(l')_{∀l'∈[l_S,N_R] | tasks at l' is schedulable} that gives the maximum cumulative tightness
    η = Σ_{τ_s∈Γ_S} η_s
    return l*, T(l*) /* return the parameters */
else
    return Unschedulable /* not possible to integrate security tasks */
end if
```

4.1 Security Applications in a Cyber-Physical System

4.1.1 Platform Overview and Customization. We implemented our ideas on a rover platform (Fig. 2) manufactured by Waveshare [48]. The rover hardware/peripherals (*e.g.,* wheel, motor, servo, sensor, etc.) are controlled by a Raspberry Pi 3 (RPi3) Model B [49] SBC (single board computer). The RPi3 is equipped with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 CPU on top of Broadcom BCM2837 SoC (system-on-chip). The base hardware unit of the rover is connected with RPi3 using a 40-pin GPIO (general-purpose input/output) header. The adapter board contains a voltage regulator (provides stable 5V power for RPi3), AD acquisition chip (to use analog sensors), servo controller (for rotating the rover) and UART converter (to control the RPi3 via UART). The base chassis is equipped with all the sensors, motor driver and micro gear motor. We also attached a camera (RPi3 camera module) that can capture 3280 × 2464 pixel static images (and also supports high-definition videos). The detailed specifications of the rover hardware (*e.g.,* base chassis, adapter, wheels, *etc.*) are available on the vendor website [48].

The RPi3 runs on a vendor-supported open-source operating system, *Raspbian* (a variant of Debian Linux). In our experiments, we used the most recent version of Raspbian (*e.g.,* Raspbian Stretch console image with Linux kernel 4.9). We enabled real-time capabilities on top of the vanilla Linux kernel by applying the PREEMPT_RT patch [50] (version 4.9.80-rt62-v7+). Since we focus on a single core platform we activated only a single core (*e.g.,* `core0`) and disabled the remaining three cores. To be specific, we modified the boot command file (`/boot/cmdline.txt`) and set the flag `maxcpus = 1`. The system configurations used in our experiments are summarized in Table 3.

4.1.2 Experimental Setup. In our experiments, the rover moved through a line, captured images and stored those in its internal storage. Our rover platform consists of the following real-time tasks (Γ_R): *(a)* four navigation tasks (for moving the rover forward, backward, left and right); *(b)* one camera task (captured still images and stored in the internal filesystem) and *(c)* one sensor logger task (for reading and logging rover's infrared sensor values). We did not make any modifications to the vendor-provided firmware/control code that ran the rover. To integrate security into this rover platform, we included additional security tasks. For the security application, we considered Tripwire

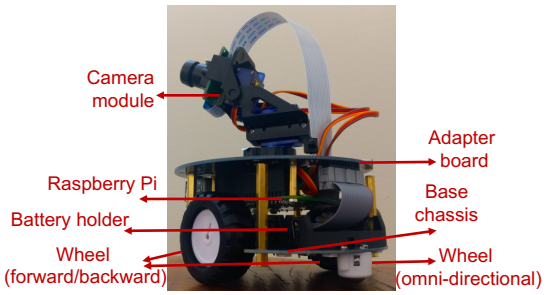

Fig. 2. Surveillance rover used in our experiments.

Table 3. Summary of the Implementation Platform

Artifact	**Configuration**
Platform	1.2 GHz 64-bit Broadcom BCM2837 (Raspberry Pi 3)
CPU	1.2 GHz 64-bit ARM Cortex-A53
Memory	1 Gigabyte
Operating System	Debian Linux (Raspbian Stretch Lite)
Kernel version	Linux Kernel 4.9
Real-time patch	PREEMPT_RT 4.9.80-rt62-v7+
Kernel flags	`CONFIG_PREEMPT_RT_FULL` enabled
Boot parameters	`maxcpus=1`, `force_turbo=1`, `arm_freq=700`, `arm_freq_min=700`

and included the following security tasks (Γ_S) that can: *(a)* protect the binary files of Tripwire; *(b)* protect system binary and *(c)* check for intrusions in the filesystem. We also modified the default configuration file of Tripwire (`/etc/tripwire/twpol.txt`) and customized it (*e.g.,* modified default rules and added new rules – refer to Tripwire manual [16] for details) for our experimental platform and rover application requirements.

We measured the execution time of real-time and security tasks using the ARM cycle counter register (CCNT) that gives us nanosecond-level precision. Since CCNT is not enabled by default in RPi3, we developed a loadable kernel module (LKM) and activated the register. We used the dual-loop timing method [14] for calculating WCETs[10], *i.e.,* we first timed an empty loop with only the measurement instrumentation and then the execution times obtained for these instrumentation-only loops were subtracted from the execution times for the loops with the task code. We set the period and control constraints in a way that the taskset $\Gamma_R \cup \Gamma_S$ became schedulable (see Tables 4 and 5). We set the cost threshold as a function of base cost $J^{B_{TH}} = \lambda \times J_r^{B_{TH}}, \forall \tau_r$ where $J_r^{B_{TH}}$ denotes the base control cost (*e.g.,* when there is no security tasks in the system) and the value of λ was varied as an experimental parameter. For this rover platform we considered the control cost as a function of task response time (*e.g.,* $\alpha_r = 0, \beta_r = 1$ and hence $J_r^{B_{TH}} = \Delta_r, \forall \tau_r \in \Gamma_R$) and assumed equal weights for the security tasks. For the accuracy of our measurements, we disabled the frequency

[10] Any existing WCET analysis technique (see the related survey [51]) can also be used with our scheme and is orthogonal to the issue at hand.

Table 4. Parameters for Real-time and Security Tasks

Real-Time Tasks	C_r (ms)	T_r (ms)	$J_r^{B_{TH}}$ (ms)
Navigation (Forward)	20.55	4111.17	205.55
Navigation (Backward)	176.43	3528.73	391.28
Navigation (Left)	147.53	2950.60	567.14
Navigation (Right)	147.64	2952.90	765.52
Camera	672.81	13456.34	1645.16
Sensor logger	98.57	1971.44	1841.41

Table 5. Parameters for Security Tasks*

Security Tasks	C_s (ms)	T_s^{des} (ms)
Scan Tripwire binary	3888.82	77776.47
Scan system binary	3926.75	58174.83
Scan filesystem	2908.74	78535.03

*$T_s^{max} = 2T_s^{des}, \forall \tau_s, l_S = \lceil 0.3 N_R \rceil$.

scaling feature in the OS. For this, we modified the boot configuration file `/boot/config.txt` (see Table 3) and allow RPi3 to execute at a constant frequency (*e.g.,* 700 MHz – the default value). We used the `GPkit` [46] library and `CVXOPT` [52] solver to solve the period selection problem.

4.1.3 Experience and Evaluation. We compared the performance of Contego-C with the approach described in our earlier research [1] where the security tasks are allowed to execute only when the real-time tasks are *not* running (*i.e.,* opportunistic execution where $l = L_s = N_R$). Note that when security tasks are executing opportunistically, they will not have any impact on the timing/control constraints of the real-time tasks.

In the first set of experiments, we analyzed the performance of our scheme by observing how quickly an intrusion can be detected. In this experiment we considered two control cost thresholds, *e.g.,* $J^{B_{TH_1}}$ and $J^{B_{TH_1}}$ – for simplicity of notation let us denote to those thresholds as TH_1 and TH_2 where $TH_1 > TH_2$ (*e.g.,* less constrained). In our experiments $TH_1 = 38 J_r^{B_{TH}}$ and $TH_2 = 35 J_r^{B_{TH}}, \forall \tau_r$ where $J_r^{B_{TH}}$ represents the base control cost (*viz.,* response time – see Table 4) for τ_r when there is no security tasks in the system.[11] For this rover platform, Algorithm 1 found higher priority-level (*e.g.,* 3) for the security tasks when the control cost is bounded by TH_1 (compared to 5 for the control cost threshold TH_2, where lower value implies higher priority). We also found that for any threshold $J^{B_{TH}} = \lambda \times J_r^{B_{TH}}$ where $1 \leq \lambda \leq 34$ our scheme returns same priority-level (*e.g.,* 6) for security tasks as opportunistic execution scheme.

Our goal here was to analyze the effectiveness of security tasks from the scheduling perspective. To illustrate malicious behavior, *viz.,* we overrode one of the real-time tasks' code and launched an attack that corrupted the logs/images collected by the rover. For each of the experiments, we

[11] **Note:** we selected these values by trial-and-error so that the optimization routine returns different priority-levels with different thresholds and we can observe the design trade-offs.

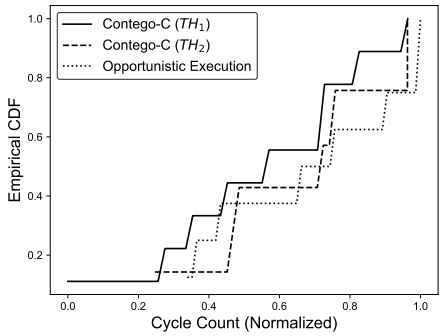

Fig. 3. Proposed scheme vs. opportunistic execution: empirical CDF of intrusion detection time. The empirical CDF is defined as $\widehat{F}_{\alpha}(\jmath) = \frac{1}{\alpha}\sum_{i=1}^{\alpha}\mathbb{I}_{[\zeta_i \leq \jmath]}$, where α is the total number of experimental observations, ζ_i is time to detect the attack in at the i-th experimental observation and $\jmath$ represents the x-axis values (*e.g.,* normalized detection time in cycle counts). The indicator function $\mathbb{I}_{[\cdot]}$ outputs 1 if the condition $[\cdot]$ is satisfied and 0 otherwise. The cost threshold TH_1 results in better tightness for this rover platform and shows a faster intrusion detection rate.

started with a clean (*i.e.,* uncompromised) system state, launched an attack at any random point of the program execution and logged the time required by Tripwire to detect the attack.[12]

The x-axis in Fig. 3 represents the normalized detection time (in cycle counts, normalized to one) and the y-axis represents the probability (*e.g.,* empirical CDF) that the attack is being detected by that time. The figure shows that Contego-C with cost threshold TH_1 provides better detection time (*e.g.,* lesser cycle count required to detect the intrusion). Since security tasks experience less interference for TH_1 we can see the faster intrusion detection rate for TH_1 compared to the other case. Note that the actual time to detect the attacks also depends on when the attack is launched and the corresponding scheduling point of the security tasks. The priority-level of security tasks is closer for both TH_2 (*e.g.,* 5 and 6 for TH_2 and opportunistic execution, respectively), and hence for TH_2 both the schemes shows similar results in terms of detection time. Since earlier work allows the security tasks to run only when other real-time tasks are not running, the feasibility region becomes more constrained and the optimization routine finds periods that are higher than those for the proposed scheme. This leads to a poorer detection rate in general in most of the experiments.

In the following experiment, we observed how our security integration approach impacted the performance of real-time tasks. We set the control cost thresholds as TH_1 so that security tasks can execute with a priority higher than the camera task.[13] For each experimental trial, we observed the schedule for 200 seconds and measured[14] the response time of the camera task (Fig. 4) using ARM cycle counter registers. We also logged the number of images captured by the task (Table 6). In Fig. 4 we show the (90^{th} percentile) response time observed in our experiments for both Contego-C and opportunistic execution scheme. As we can see from the figure, task response time increases for Contego-C. Since our scheme allows security tasks to execute with a priority higher than some

[12]We assumed that there are no zero-day attacks and that attacks are detected by Tripwire correctly (*i.e.,* there are no false positive/negative errors).

[13]We show the results for a case where security tasks execute with higher priority than the camera task. Otherwise, both Contego-C and opportunistic execution scheme will have the same impact on the camera task – since they will not cause any interference to the camera task.

[14]We took measurements for demonstration purposes only. Such runtime probing may negatively impact the real-time performance for practical use-cases.

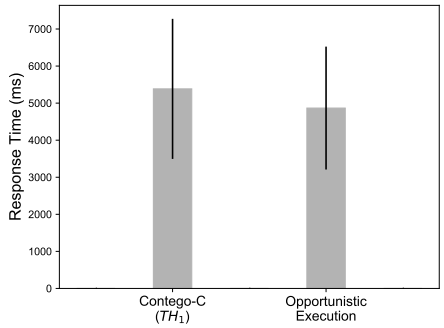

Fig. 4. Contego-C vs. opportunistic execution: impact on the response time of a low-priority real-time task (*e.g.,* camera task). Promoting the priority of the security tasks increases the response time of the camera task.

real-time tasks (camera task[15] in this case), the low-priority real-time tasks will suffer interference from security tasks. As a result response time increases. From our experiment, we found that on average the response time of the surveillance task increased by 13.60%. However as we can see from Table 6, this increased response time did not degrade system throughput (in terms of the number of images captured by the surveillance task for a given duration) – since on average both schemes captured an equal number of images in our experimental trials.

Table 6. Number of Images Captured by Both the Schemes

	Image Statistics*	
Scheme	**Avg.**	**S.D.**
Contego-C (TH_1)	2	0.54
Opportunistic Execution	2	0.40

*Statistics over 15 trials (each ran for 200 seconds).

4.2 Experiments with Synthetic Tasksets

4.2.1 Workload Generation and Parameters. We used the parameters similar to that from prior research [1, 43, 53]. We grouped the real-time and security tasksets by base-utilization from $[0.01 + 0.1 \cdot i, 0.1 + 0.1 \cdot i]$ where $i \in \mathbb{Z}, 0 \leq i \leq 9$. This allowed us to generate tasksets with an even distribution of tasks. Each base-utilization group contained 250 tasksets (*e.g.,* a total of 2500 tasksets were tested for each of the experiments). The utilization of the real-time and security tasks were generated by the UUniFast [54] algorithm. Each taskset instance contained $[3, 10]$ real-time and $[2, 5]$ security tasks. Each real-time task $\tau_j \in \Gamma_R$ had a period $T_j \in [10, 1000]$ ms and we assumed $l_S = \lceil 0.3 N_R \rceil$. The maximum allowable periods for the security tasks were selected from $[1000, 1500]$ ms and the desired period was assumed to be $T_s^{des} = \lfloor 0.5 T_s^{max} \rfloor$, $\forall \tau_s \in \Gamma_S$. We also assumed that real-time task priorities follow rate-monotonic order [28], *e.g.,* priority of τ_r is higher than $\tau_{r'}$ if $T_r < T_{r'}$. For security tasks, we assumed that priorities are assigned according to desired monitoring frequency, *e.g.,* priority of τ_s is higher than $\tau_{s'}$ if $T_s^{des} < T_{s'}^{des}$.

[15]We considered camera task as an example of low-priority real-time task to demonstrate the effects.

Table 7. Control Systems and Parameters

Control System	Transfer Function*	Parameters
Cruise control	$\frac{1}{1500s+50}$	$\alpha = \frac{5.57}{10^6}$ $\beta = \frac{5.46}{10^6}$
Rotary motion control	$\frac{0.01}{(0.01s+0.1)(0.5s+1)+(0.01)^2}$	$\alpha = \frac{695}{10^4}$ $\beta = \frac{682}{10^4}$
Suspension system	$\frac{0.0035s^2+0.01876s+0.625}{s^4+48.1s^3+1849s^2+1657s+5\times10^4}$	$\alpha = \frac{7.34}{10^9}$ $\beta = \frac{7.20}{10^9}$

*Given the transfer function $G(s)$, it is straightforward to correlate with the state space form (*e.g.,* Eq. (1)). For details refer to [55, Ch. 2].

Table 8. Simulation Parameters

Parameter	Values
Number of real-time tasks, N_R	$[3, 10]$
Number of security tasks, N_S	$[2, 5]$
Real-time task period, T_r	$[10, 1000]$ ms
Maximum allowable period, T_s^{max}	$[1000, 1500]$ ms
Desired period for security tasks, T_s^{des}	$\lfloor 0.5T_s^{max} \rfloor$
Minimum utilization of security tasks	30% of real-time tasks
Maximum priority-level of the security tasks, l_S	$\lceil 0.3N_R \rceil$
Number of taskset in each configuration	250

For modeling physical plants, we considered three (linearized) automotive control systems [56]: *(a)* cruise control system (operates the vehicle at a constant speed), *(b)* rotary control system (coupled with wheels or drums and provide translational motion) and *(c)* vehicle suspension system (single dimensional multiple spring-damper system that controls the motion of the vehicle body). For each of these three control systems P_i, we first used Jitterbug tool [57] to obtain the quadratic control cost J_i (*e.g.,* Eq. (2)) and then linearized and obtained the parameters α_i and β_i using `scipy.optimize` [58] library (see Table 7). For each of the real-time tasks $\tau_r \in \Gamma_R$, we randomly selected the control parameters from $\{\alpha_i\}, \{\beta_i\}$, $1 \leq i \leq 3$. Unless otherwise specified, for low-priority real-time tasks we set the control cost threshold $J_r^{TH} = 5J_r^{B_{TH}}$ where $J_r^{B_{TH}}$ represents the base cost (*e.g.,* the cost when there is no security tasks or the security tasks are executing with lowest priority). We considered $\omega_s = 1$, $\forall \tau_s \in \Gamma_S$ and the total utilization of the security tasks were assumed to be at least 30% of the real-time tasks. The parameters used in our experiments are summarized in Table 8.

4.2.2 Comparison With Opportunistic Execution. In the first set of experiments (Fig. 5) we compare the number of schedulable tasksets (*e.g.,* those ones where all the real-time requirements are satisfied) found by both the proposed and opportunistic execution schemes. We used the *acceptance ratio* as a metric to evaluate schedulability. The acceptance ratio (y-axis in Fig. 5) is defined as the number of schedulable tasksets over the total number of generated ones. The x-axis in Fig. 5 shows the total system utilization: $\sum_{\tau_i \in \{\Gamma_R \cup \Gamma_S\}} \frac{C_i}{T_i}$ (*e.g.,* total utilization of real-time and security tasks). From this figure we can observe that Contego-C results in better schedulability compared to *opportunistic execution* scheme. Our proposed scheme allows security tasks to execute at a priority-level up to l_S (as long as it does not violate control delay requirements). As a result, security tasks experience less interference than in *opportunistic execution* scheme. This flexibility gives the optimization routine

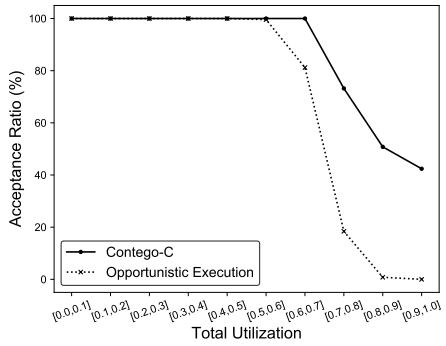

Fig. 5. Percentage of tasksets found schedulable (*e.g.,* that satisfied all the constraints) in both schemes for different base-utilizations.

a larger feasibility region (especially for high utilization) to satisfy all the constraints (and hence more tasksets are found schedulable).

In the next set of experiments, we analyzed how the different control cost impacts the schedulability. For this, we vary the cost threshold as a function of base cost: $J^{B_{TH}} = \lambda \times J_r^{B_{TH}}$, $\forall \tau_r$ and show how it impacts schedulability for different utilizations and base cost factors (*e.g.,* λ). For better representation of schedulability with different utilizations and cost thresholds we define the *weighted schedulability* metric as follows [59]:

$$W_s(J^{TH}) = \frac{\sum_{\Gamma} U(\Gamma) S(\Gamma, J^{TH})}{\sum_{\Gamma} U(\Gamma)} \tag{26}$$

where schedulability test $S(\Gamma, J^{TH})$ returns a binary output if the taskset Γ is schedulable for cost threshold J^{TH} and $U(\Gamma)$ denotes the taskset utilization in all base-utilization groups. The x-axis in Fig. 6 shows the different cost threshold (as a factor of base cost) and the y-axis shows the weighted schedulability. From this figure, we can see that a higher cost threshold increases schedulability since the optimization method finds a wider feasibility region to satisfy all the real-time constraints. Also, the opportunistic execution scheme does not have any impact on the real-time tasks, and thus weighted schedulability remains unchanged with varying cost thresholds.

In Fig. 7 we measure the difference in the tightness of periodic monitoring (*e.g.,* η) obtained by the proposed scheme and the opportunistic execution approach. The non-negative values in the y-axis of Fig. 7 imply that the proposed scheme performs better than the scheme presented in our previous work [1]. The figure shows that our approach can achieve better cumulative tightness, and performs comparatively better in medium utilization (*e.g.,* 0.45-0.75). For higher utilization, the difference is close to zero. However, this does not mean that the proposed scheme performs worse than the opportunistic execution approach. This is mainly because our scheme found a feasible solution for the lowest priority-level (*e.g.,* $l = N_R$) and is making both schemes look more similar.

4.2.3 Comparison with Mixed-Criticality Systems and Optimal Priority Ordering. While in this work we consider a legacy system (*i.e.,* where changing the priority ordering of all real-time tasks is not an option), for comparison purposes we considered the following two schemes (Note: they do not consider any period adaptation):

- *Criticality Monotonic Priority Ordering (CrMPO)* [26]: We compare with mixed criticality systems [27] where tasks have different criticality levels (*e.g.,* HI and LO) to ensure the level

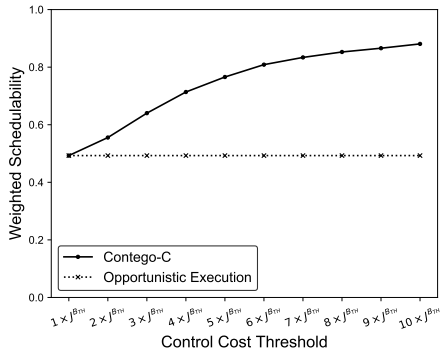

Fig. 6. Weighted schedulability with different control cost thresholds. Schedulability of opportunistic execution scheme is independent of cost threshold.

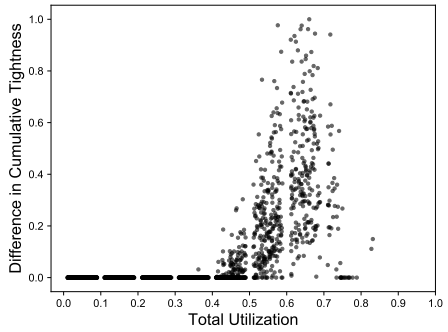

Fig. 7. The difference in cumulative tightness for Contego-C and opportunistic execution. Each of the data points in the plot represents a schedulable taskset (*e.g.,* there exists a solution that satisfies all the constraints).

of assurance. In our experiments tasks follow a criticality monotonic ordering (HI-criticality tasks have higher priorities than all the LO-criticality tasks), *e.g.,* first l_S real-time and all security tasks are considered as HI-criticality and the rest of $N_R - l_S$ real-time tasks are scheduled as LO-criticality. Since we focus on schedulability, we consider two variants of CrMPO: CrMPO-Tmax and CrMPO-Tdes where periods of the security tasks are set as T_s^{max} and T_s^{des}, $\forall \tau_s \in \Gamma_S$, respectively.

- *Optimal Priority Assignment (OPA)* [60]: In this scheme, all the tasks (*e.g.,* both real-time and the security ones) are scheduled using Audsley's OPA algorithm. As before, we consider the two variants of OPA: OPA-Tmax and OPA-Tdes where periods of the security tasks are set to maximum and desired values, respectively.

In Fig 8 we compare the acceptance ratio of Contego-C with CrMPO and OPA schemes. We note that any taskset that is schedulable by CrMPO-Tmax/CrMPO-Tdes schemes will also be schedulable in Contego-C. Running all the security tasks with their desired periods will result in a high degree of interference from low priority tasks and this leads to a poor acceptance ratio (especially for higher utilization) for CrMPO-Tdes scheme. While both the OPA-Tmax and OPA-Tdes outperform Contego-C (for higher utilizations), we note that this may not be applicable for legacy systems where changing the priority of (all or a subset of) real-time/control tasks is not an option. We also

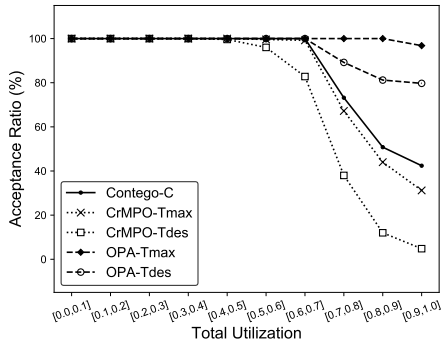

Fig. 8. The acceptance ratio vs taskset utilizations for Contego-C and other priority assignment schemes (*e.g.,* CrMPO and OPA).

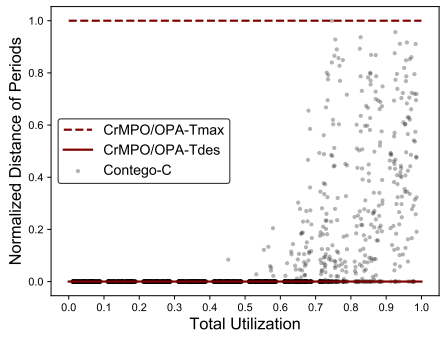

Fig. 9. Normalized difference between achievable and desired periods for each of the security tasks vs. total utilization of the system. The closer the y-axis values to 0, the more desirable the period for security tasks.

note that while OPA-Tmax results in better schedulability, our priority and period assignments help to achieve faster monitoring – as we explain next (see Fig. 9).

The cumulative tightness η bound presented in Section 4.2.2 takes into account all the security tasks. In the following experiment (Fig. 9), we measure how close the execution frequency of *each* of the security tasks is to their desired frequency when compared to the CrMPO/OPA schemes (red lines in the figure). Let us define the following metric (*e.g.,* normalized difference between achievable and desired periods):

$$\xi = \frac{\|\mathbf{T}^* - \mathbf{T}^{\mathbf{des}}\|_2}{\|\mathbf{T}^{\mathbf{max}} - \mathbf{T}^{\mathbf{des}}\|_2} \tag{27}$$

where $\mathbf{T}^*$ is the solution obtained from Algorithm 1, $\mathbf{T}^{\mathbf{des}} = [T_i^{des}]^{\mathsf{T}}_{\forall \tau_i \in \Gamma_S}$ and $\mathbf{T}^{\mathbf{max}} = [T_i^{max}]^{\mathsf{T}}_{\forall \tau_i \in \Gamma_S}$ are the desired and maximum period vector respectively, and $\|\cdot\|_2$ denotes the Euclidean norm. The closer the value of ξ is to 0, the nearer the period of each of the security task is to the desired period. Note that CrMPO/OPA-Tdes (respectively CrMPO/OPA-Tmax) gives the bound of the normalized distance, *e.g.,* $\xi_{\text{CrMPO/OPA-Tdes}} = 0$ (res. $\xi_{\text{CrMPO/OPA-Tmax}} = 1$). As the total utilization increases, the feasible set of the period selection problem (that respects all constraints) becomes more restrictive due to the higher interference. As a result, we see the degradation in effectiveness (in terms of ξ) for the tasksets with higher utilization. Our experiments also show that there is a trade-off between security and schedulability. When compared to CrMPO-Tmax/OPA-Tmax our approach

finds smaller periods (since the normalized distance is less than 1) in most cases – this is expected since there is no period adaptation. Our approach achieves better monitoring when compared with CrMPO/OPA with *(b)* same or better schedulability (for CrMPO-Tmax) and *(b)* a reduction in acceptance ratio (for OPA-Tmax).

5 RELATED WORK

Security in RTS has been addressed in literature in different contexts – in a broader sense this includes (but is not limited to) integrating monitoring and intrusion detection mechanisms, protecting communication channels, defending against side-channel attacks, as well as designing hardware/software-based architectural solutions.

Real-Time Security Integration Frameworks. The work most closely related to Contego-C is our earlier work [1], where security tasks are executed with the lowest priority relative to real-time tasks. Although this approach may be useful for existing systems since the schedulability of the real-time tasks remain unaffected, as we observed from our experiments, this leads to longer response times for the security tasks and thus may increase the detection time for attacks. We also developed a multi-mode framework [12] that allows the security policies/tasks to execute in different modes (*i.e.,* passive monitoring with lowest priority as well as exhaustive checking with higher priority). By using this approach, for instance, security routines can execute opportunistically when the system is deemed to be clean (*i.e.,* not compromised). However if any anomaly or unusual behavior is suspected, the security policy will switch to a *fine-grained checking mode* and execute with higher priority. The security tasks may go back to normal mode if: *i*) no anomalous activity is found; or *ii*) the intrusion is detected and malicious entities are removed. Our multi-mode framework [12] used the concept of hierarchical scheduling and proposed to execute the security tasks in a *server* [61]. Such server-based approach is difficult to implement in practical systems and requires additional porting efforts. In recent work [62] we introduced algorithm to find an allocation of security tasks for partitioned fixed-priority multicore RTS [63] using the concept of opportunistic execution. All of the aforementioned work do *not* consider the control aspects and are designed for implicit-deadline systems only.

Mixed Criticality Scheduling and Priority/Period Optimization in RTS. The system model considered in this paper may be viewed as a special case of mixed criticality systems [64] where the system operates in multiple criticality levels (say HI for real-time and LO for security tasks). However, unlike Vestal's mixed criticality task model [27] where WCETs and periods are vectors of values (see the related survey [64]), we consider a single WCET and period value (*e.g.,* there exists only one criticality level). In mixed criticality systems LO criticality tasks are abandoned to ensure timely operation of the HI-criticality tasks [65] that may not be possible in our context. In addition, as we see in the experiments (Section 4.2.3) generic criticality monotonic priority ordering [26] does not always result in better monitoring (*e.g.,* less effective). We further note that while there exist a large body of work in mixed criticality RTS (too many to enumerate here, refer to related surveys [64, 66–68]), the consideration of real-time security aspects (*i.e.,* joint period/priority selection of security tasks) distinguish our research from existing literature.

Researchers also develop various priority assignment schemes for RTS (see the related work [69–72] and the survey [65]). However, existing work *(a)* focus on assigning priorities of *all* the tasks, *(b)* are not control-aware, *(c)* do not optimize task periods and *(d)* are primarily designed for implicit-deadline systems (*i.e.,* deadline $\leq$ period). Further, they are also not designed toward security considerations of legacy systems in mind where real-time task parameters can not be significantly modified due to real-time/control constraints.

While not in the context of security in RTS, there exists other work [32, 73] in which the authors statically assign the periods for multiple independent control tasks considering control delay as a cost metric. Davare *et al.* [74] propose to assign task and message periods as well as satisfy end-to-end latency constraints for distributed automotive systems by leveraging schedulability analysis within a convex optimization framework. Previous work used a different model/application scenario (such as controller area networks [75] and/or minimize control delay using utilization-bound tests) and hence can not be directly retrofitted in our context.

Securing Communication Messages/Channels. There exists recent work [10, 11] where authors proposed schemes to secure cyber-physical systems from man-in-the-middle attacks, where an attacker can compromise communication between system sensors and controllers. There has been some work [8, 9] where authors proposed to add security mechanisms (such as encryption) into RTS and considered periodic task scheduling where each task requires a security service whose overhead varies according to the quantifiable level of the service. Unlike ours, all of the aforementioned work *require modification of the existing real-time tasks.*

Defense Against Side-Channel Attacks. Bao *et al.* [76] model the behavior of the attacker and introduce a scheduling algorithm. Unlike hard real-time systems, authors consider a system with aperiodic tasks that have *soft* deadlines. The proposed polynomial complexity algorithm provides a trade-off between side-channel information leakage and the number of deadline misses for the real-time tasks. In comparison, we propose to ensure security policies in hard RTS *without* violating temporal constraints and schedulability of the real-time (and control) tasks. A state cleanup mechanism is introduced in literature [38] where the authors modify the fixed priority scheduling algorithm to mitigate information leakage through shared resources (*e.g.,* caches). However, this leakage prevention comes at a cost of reduced schedulability. In comparison, we propose to ensure security policies *without* violating temporal constraints and schedulability of the real-time control tasks.

Randomization and Architectural Frameworks. Researchers also proposed schedule obfuscation methods [77] to minimize the predictability of deterministic RTS scheduler by randomizing the task schedule while providing the necessary real-time guarantees. Unlike our scheme that works at the scheduler level, there exist architectural frameworks [13, 14, 24, 25, 78–81] that can protect RTS against security vulnerabilities. We highlight that all the aforementioned work requires modification to the scheduler or real-time task parameters. It is not inconceivable that those architectural frameworks and randomization protocols can the employed on top of our proposed scheme to improve security posture in future RTS.

6 DISCUSSION

In this work, we did not design Contego-C towards any specific security mechanisms and allow designers to integrate any given technique based on application requirements. Depending on the actual operation of the security tasks a particular (class of) attack may or may not be detectable. For instance Contego-C may not detect a zero-day exploit for a given set of security tasks.

There exist cases when some of the security tasks may need to be executed *without preemption.* For instance, let us consider a security task that scans the process table and has been preempted in the middle of its operation. An adversary may corrupt the process table entry that has already been scanned before the next scheduling point of the security task. When the security tasks are rescheduled, it will start scanning from its last known state and may not be able to detect the changes in a timely manner. When security tasks need to perform a special atomic operation, the priority of the task can be increased to a priority that is strictly higher than all of the real-time tasks.

It is worth mentioning that, the cost of atomicity (by means of priority inversion) will compromise the timing constraints (and also control performance) of some (or all) of the real-time tasks.

While Contego-C abstracts security tasks (and underlying monitoring events) and works in a proactive manner, designers may want to integrate monitoring mechanisms that *react*, based on anomalous behavior. For instance, let at time t, j-th job of task τ_s (*e.g.,* τ_s^j) performs action a_0 (*e.g.,* runtime of real-time tasks). Because of intrusions (or perhaps due to other system artifacts) in time $[t, t + T_s]$ (T_s is the period of τ_s), job τ_s^{j+1} finds that a_0 is not behaving as expected. Therefore τ_s^{j+1} may perform both actions, a_0 and a_1 (say that checks the list of system calls, to see if any undesired calls are executed). One way to support such a feature is to consider the dependency (*i.e.,* a_1 depends on a_0 in this case) between security checks (*e.g.,* sub-tasks). We intend to extend our framework considering dependencies between security tasks.

7 CONCLUSION

Any successful security breach in cyber-physical systems with real-time requirements can have catastrophic effects. Threats to safety-critical systems are growing and there is a need to design security solutions that can foil such attacks. In this paper, we introduce Contego-C, a framework to integrate security into legacy real-time control systems. We demonstrate the efficacy of such integration mechanisms in a practical cyber-physical system and analyze the design trade-offs, both from a security and real-time perspective. We believe Contego-C will provide valuable hints to the engineers on how to enhance security into such safety-critical systems.

REFERENCES

[1] M. Hasan, S. Mohan, R. B. Bobba, and R. Pellizzoni, "Exploring opportunistic execution for integrating security into legacy hard real-time systems," in *IEEE RTSS*, 2016, pp. 123–134.

[2] M. Abrams and J. Weiss, "Malicious control system cyber security attack case study–Maroochy water services, Australia," *McLean, VA: The MITRE Corporation*, 2008.

[3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX Sec. Symp.*, 2011.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *IEEE S&P*, 2010, pp. 447–462.

[5] S. S. Clark and K. Fu, "Recent results in computer security for medical devices," in *MobiHealth*, 2011, pp. 111–118.

[6] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, p. 6, 2011.

[7] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the ukrainian power grid," *SANS Industrial Control Systems*, 2016.

[8] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," *IEEE Trans. on Indust. Info.*, vol. 5, no. 1, pp. 22–37, 2009.

[9] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," *ACM TECS*, vol. 6, no. 3, p. 20, 2007.

[10] V. Lesi, I. Jovanov, and M. Pajic, "Network scheduling for secure cyber-physical systems," in *IEEE RTSS*, 2017, pp. 45–55.

[11] V. Lesi, I. Jovanov, and M. Pajic, "Security-aware scheduling of embedded control tasks," *ACM TECS*, vol. 16, pp. 188:1–188:21, 2017.

[12] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba, "Contego: An adaptive framework for integrating security tasks in real-time systems," in *Euromicro ECRTS*, 2017, pp. 23:1–23:22.

[13] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, and L. Sha, "SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems," in *IEEE RTAS*, 2013, pp. 21–32.

[14] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "S3A: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems," in *ACM international conference on High confidence networked systems*. ACM, 2013, pp. 65–74.

[15] J. Song, G. Fry, C. Wu, and G. Parmer, "CAML: Machine learning-based predictable system-level anomaly detection," in *IEEE CERTS*, 2016, pp. 12–18.

[16] "Tripwire," https://github.com/Tripwire/tripwire-open-source.

[17] "Advanced Intrusion Detection Environment (AIDE)," http://aide.sourceforge.net/.
[18] "The Bro network security monitor," https://www.bro.org.
[19] M. Roesch, "Snort - lightweight intrusion detection for networks," in *USENIX Conf. on Sys. Admin.*, 1999, pp. 229–238.
[20] L. L. Woo, M. Zwolinski, and B. Halak, "Early detection of system-level anomalous behaviour using hardware performance counters," in *DATE*, 2018, pp. 485–490.
[21] V. M. Weaver, "Linux perf_event features and overhead," in *IEEE FastPath*, 2013.
[22] "OProfile," http://oprofile.sourceforge.net/.
[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM CSUR*, vol. 41, no. 3, p. 15, 2009.
[24] M.-K. Yoon, S. Mohan, J. Choi, and L. Sha, "Memory heat map: anomaly detection in real-time embedded systems using memory behavior," in *ACM/EDAC/IEEE DAC*, 2015, pp. 1–6.
[25] M.-K. Yoon, S. Mohan, J. Choi, M. Christodorescu, and L. Sha, "Learning execution contexts from system call distribution for anomaly detection in smart embedded system," in *ACM/IEEE IoTDI*, 2017, pp. 191–196.
[26] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *IEEE RTSS*, 2011, pp. 34–43.
[27] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *IEEE RTSS*, 2007, pp. 239–243.
[28] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.
[29] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *RTS Journal*, vol. 6, no. 2, pp. 133–151, 1994.
[30] E. Todorov, "Optimal control theory," *Bayesian brain: probabilistic approaches to neural coding*, pp. 269–298, 2006.
[31] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-feedforward scheduling of control tasks," *RTS Journal*, vol. 23, no. 1-2, pp. 25–53, 2002.
[32] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *IEEE RTSS*, 2008, pp. 291–300.
[33] Y. Xu, K.-E. Årzén, E. Bini, and A. Cervin, "Response time driven design of control systems," *IFAC Proceedings*, vol. 47, no. 3, pp. 6098–6104, 2014.
[34] Y. Wu, G. Buttazzo, E. Bini, and A. Cervin, "Parameter selection for real-time controllers in resource-constrained systems," *IEEE Trans. on Ind. Info.*, vol. 6, no. 4, pp. 610–620, 2010.
[35] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *IEEE RTSS*, 1990, pp. 201–209.
[36] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.
[37] X. Wang, N. Hovakimyan, and L. Sha, "L1Simplex: Fault-tolerant control of cyber-physical systems," in *2013 ACM/IEEE ICCPS*, 2013, pp. 41–50.
[38] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Integrating security constraints into fixed priority real-time schedulers," *RTS Journal*, vol. 52, no. 5, pp. 644–674, 2016.
[39] X. Zhang, J. Zhan, W. Jiang, Y. Ma, and K. Jiang, "Design optimization of security-sensitive mixed-criticality real-time embedded systems," in *IEEE ReTiMiCS*, 2013.
[40] K. Jiang, P. Eles, and Z. Peng, "Optimization of secure embedded systems with dynamic task sets," in *DATE*, 2013, pp. 1765–1770.
[41] S. Mohan, "Worst-case execution time analysis of security policies for deeply embedded real-time systems," *ACM SIGBED Review*, vol. 5, no. 1, p. 8, 2008.
[42] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *SE Journal*, vol. 8, no. 5, pp. 284–292, 1993.
[43] M.-K. Yoon, J.-E. Kim, R. Bradford, and L. Sha, "Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling," in *DATE*, 2013, pp. 1313–1318.
[44] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Opt. & Eng.*, vol. 8, no. 1, pp. 67–127, 2007.
[45] S. Boyd and L. Vandenberghe, *Convex optimization*, 2004.
[46] E. Burnell and W. Hoburg, "GPkit software for geometric programming," 2017. [Online]. Available: https://github.com/hoburg/gpkit
[47] A. Mutapcic, K. Koh, S. Kim, L. Vandenberghe, and S. Boyd, "GGPLAB: a simple Matlab toolbox for geometric programming," 2006. [Online]. Available: https://stanford.edu/~boyd/ggplab/
[48] "Alphabot2 wiki," https://www.waveshare.com/wiki/AlphaBot2-Pi.
[49] "Raspberry Pi 3 Model B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.
[50] L. Fu and R. Schwebel, "Real-time Linux wiki," https://rt.wiki.kernel.org/index.php/rt_preempt_howto, [Online].
[51] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM TECS*, vol. 7, no. 3, p. 36, 2008.

[52] L. Vandenberghe, "The CVXOPT linear and quadratic cone program solvers," 2010. [Online]. Available: http://cvxopt.org/documentation/coneprog.pdf
[53] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Real-time systems security through scheduler constraints," in *Euromicro ECRTS*, 2014, pp. 129–140.
[54] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *RTS Journal*, vol. 30, no. 1-2, pp. 129–154, 2005.
[55] K. Ogata, *Modern control engineering*, 5th ed. Prentice hall, 2010.
[56] "Control tutorials for MATLAB and Simulink," http://ctms.engin.umich.edu/CTMS/.
[57] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *IEEE CDC*, vol. 2, 2002, pp. 1319–1324.
[58] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: open source scientific tools for Python." [Online]. Available: http://www.scipy.org/
[59] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *OSPERT*, pp. 33–44, 2010.
[60] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Elsevier Inf. Proc. Letters*, vol. 79, no. 1, pp. 39–44, 2001.
[61] R. Davis and A. Burns, "An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems," in *IEEE RTNS*, 2008.
[62] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba, "A design-space exploration for allocating security tasks in multicore real-time systems," in *DATE*, 2018, pp. 225–230.
[63] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, 2011.
[64] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM CSUR*, vol. 50, no. 6, p. 82, 2018.
[65] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *Elsevier J. of sys. arch.*, vol. 65, pp. 64–82, 2016.
[66] J. Simó, P. Balbastre, J. F. Blanes, J.-L. Poza-Luján, and A. Guasque, "The role of mixed criticality technology in industry 4.0," *Electronics*, vol. 10, no. 3, p. 226, 2021.
[67] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *ACM RTNS*, 2015, pp. 139–148.
[68] H. Chai, G. Zhang, J. Sun, A. Vajdi, J. Hua, and J. Zhou, "A review of recent techniques in mixed-criticality systems," *J. of Cir., Sys. and Comp.*, vol. 28, no. 07, p. 1930007, 2019.
[69] G. de A Lima and A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," *IEEE TC*, vol. 52, no. 10, pp. 1332–1346, 2003.
[70] Y. Zhao and H. Zeng, "The concept of unschedulability core for optimizing priority assignment in real-time systems," in *IEEE DATE*, 2017, pp. 232–237.
[71] A. LCTES, "Priority assignment for embedded reactive real-time systems," in *Languages, Compilers, and Tools for Embedded Systems*, 1998, pp. 146–155.
[72] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *IEEE RTSS*, 2007, pp. 3–14.
[73] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, "Control-quality driven design of cyber-physical systems with robustness guarantees," in *DATE*, 2013, pp. 1093–1098.
[74] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *ACM DAC*, 2007, pp. 278–283.
[75] K. Tindell, H. Hanssmon, and A. J. Wellings, "Analysing real-time communications: Controller area network (CAN)." in *IEEE RTSS*, 1994, pp. 259–263.
[76] C. Bao and A. Srivastava, "A secure algorithm for task scheduling against side-channel attacks," in *International Workshop on Trustworthy Embedded Devices*. ACM, 2014, pp. 3–12.
[77] 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 *IEEE RTAS*, 2016, pp. 1–12.
[78] D. Lo, M. Ismail, T. Chen, and G. E. Suh, "Slack-aware opportunistic monitoring for real-time systems," in *IEEE RTAS*, 2014, pp. 203–214.
[79] F. Abdi, M. Hasan, S. Mohan, D. Agarwal, and M. Caccamo, "ReSecure: A restart-based security protocol for tightly actuated hard real-time systems," in *IEEE CERTS*, 2016, pp. 47–54.
[80] F. Abdi, J. V. D. Woude, Y. Lu, S. Bak, M. Caccamo, L. Sha, R. Mancuso, and S. Mohan, "On-chip control flow integrity check for real time embedded systems," in *IEEE CPSNA*, 2013, pp. 26–31.
[81] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *ACM/IEEE ICCPS*, 2018, pp. 10–21.