Contego: An Adaptive Framework for Integrating Security Tasks in Real-Time Systems

Monowar Hasan[1], Sibin Mohan[2], Rodolfo Pellizzoni[3], and Rakesh B. Bobba[4]

1 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
mhasan11@illinois.edu
2 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
sibin@illinois.edu
3 **University of Waterloo, Ontario, Canada**
rodolfo.pellizzoni@uwaterloo.ca
4 **Oregon State University, Corvallis, OR, USA**
rakesh.bobba@oregonstate.edu

Abstract

Embedded real-time systems (RTS) are pervasive. Many modern RTS are exposed to unknown security flaws, and threats to RTS are growing in both number and sophistication. However, until recently, cyber-security considerations were an afterthought in the design of such systems. Any security mechanisms integrated into RTS must (a) *co-exist* with the real-time tasks in the system and (b) operate *without* impacting the timing and safety constraints of the control logic. We introduce Contego, an approach to integrating security tasks into RTS without affecting temporal requirements. Contego is specifically designed for *legacy* systems, *viz.,* the real-time control systems in which major alterations of the system parameters for constituent tasks is not always feasible. Contego combines the concept of *opportunistic execution* with hierarchical scheduling to maintain compatibility with legacy systems while still providing flexibility by allowing security tasks to operate in different *modes*. We also define a metric to measure the effectiveness of such integration. We evaluate Contego using synthetic workloads as well as with an implementation on a realistic embedded platform (an open-source ARM CPU running real-time Linux).

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems

Keywords and phrases real-time systems, security, hierarchical scheduling

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.23

1 Introduction

Embedded real-time systems (RTS) are used to monitor and control physical systems and processes in many domains, *e.g.,* manned and unmanned vehicles including aircraft, spacecraft, unmanned aerial vehicles (UAVs), submarines and self-driving cars, critical infrastructures like the electric grid and process control systems in industrial plants, to name just a few. They rely on a variety of inputs for correct operation and have to meet stringent safety and timing requirements. Failures in RTS can have catastrophic consequences for the environment, the system, and/or human safety [1, 10].

Traditionally, RTS were designed using proprietary protocols, platforms and software and were not connected to the rest of the world, *i.e.,* they were air gapped. As a result cyber-security was not a design priority in such systems. However, the drive towards remote monitoring and control facilitated by the growth of the Internet, the rise in the use of

commercial-off-the-shelf (COTS) components, standardized communication protocols and the high value of these systems to adversaries have been challenging the status quo. While safety and fault-tolerance have long been important design considerations in such systems, traditional fault-tolerance techniques that were designed to counter and survive random or accidental faults are not sufficient to deal with cyber-attacks orchestrated by an intelligent and capable adversary. A number of high-profile attacks on real systems, *e.g.,* Stuxnet [15] and attack demonstrations by researchers on automobiles [20, 10] and medical devices [12] have shown that the threat is real.

Given the increasing cyber-security risks, it is essential to have a layered defense and integrate resilience against such attacks into the design of controllers and actuators (*i.e.,* embedded RTS). It is also critical to retrofit existing controllers and actuators with protection, detection, survival and recovery mechanisms. However, *any security mechanisms have to co-exist with real-time tasks in the system and have to operate without impacting the timing and safety constraints of the control logic.* This creates an apparent tension between security requirements (*e.g.,* having enough cycles for effective monitoring and detection) and the timing and safety requirements. For example, how often and how long should a monitoring and detection task run to be effective but not interfere with real-time control or other safety-critical tasks? While this tension could potentially be addressed for newer systems at design time, it is especially challenging in the retrofitting of *legacy* systems for which the control tasks are already in place and perhaps *cannot be modified.* Another challenge is to ensure that an adversary cannot easily evade such mechanisms. Further, the deterministic nature of task schedules in RTS may provide attackers with known windows of opportunity in which they can run undetected [11, 41].

Our focus in this work is on *retrofitting security mechanisms into legacy RTS*, for which modification of existing real-time tasks' parameters (such as run-times, period, task execution order, *etc.*) is not always feasible. In contrast to existing mechanisms [24, 42], the proposed method does *not* require any architectural modifications and hence is particularity suitable for systems designed using COTS components. The framework developed in this paper is based on our earlier work [18] in which we proposed to incorporate monitoring and detection mechanisms by implementing them as separate *sporadic tasks* and executing them *opportunistically*, that is, with the lowest priority so that real-time tasks are not affected. However, if the security tasks always execute with lowest priority, they suffer more interference (*i.e.,* preemption from high-priority real-time tasks) and the consequent longer detection time (due to poor response time) will make the security mechanisms less effective. In order to provide *better responsiveness* and increase the effectiveness of monitoring and detection mechanisms, we now propose a multi-mode framework called Contego[1]. For the most part, Contego executes in a PASSIVE mode with opportunistic execution of intrusion detection tasks as before [18]. However, Contego will *switch to an* ACTIVE *mode of operation* to perform additional checks as needed (*e.g.,* fine-grained analysis, used as an example in Section 6.2). This ACTIVE mode potentially executes with higher priority, while ensuring the schedulability of real-time tasks. Thus Contego subsumes the approach in our earlier work [18] and provides faster detection.

The contributions of this paper can be summarized as follows:

- We introduce Contego, an extensible framework to integrate security tasks into legacy RTS (Section 2).
- Contego allows the security tasks to execute with minimal perturbation of the scheduling

[1] A preliminary version [19] of this work was presented at a workshop without published proceedings.

order of the real-time tasks while guaranteeing their timing constraints (Sections 4–5). The proposed method can adapt to changes due to malicious activities by switching its mode of operation.

- We propose a metric to measure the security posture of the system in terms of frequency of execution (Section 3).
- We evaluate the schedulability and security of the proposed approach using a range of synthetic task sets and a prototype implementation on an ARM-based development board with real-time Linux (Section 6).

2 Security and System Model

2.1 Attack Model

RTS face threats in various forms, depending on the system and the goals of an adversary. For example, adversaries may insert, eavesdrop on or modify messages exchanged by system components, may manipulate the processing of sensor inputs and actuator commands and/or could try to modify the control flow of the system [42]. Further, rather than try to crash the system aggressively, an intruder in reconnaissance mode may want to monitor the system behavior and gather information for later use. For instance, an intruder may utilize side-channels to monitor the system behavior and infer system information (*e.g.,* hardware/software architecture, user tasks and thermal profiles, *etc.*) that may eventually help maximize the impact of an attack [11].

Let us consider an RTS (say an avionics electronic control unit) developed using a multi-vendor model [30], *viz.,* its components are manufactured and integrated by different vendors. For example, tasks in the system component manufactured by vendor v_i are very sensitive and considered classified or mission-critical (*e.g.,* images captured by the camera on the surveillance UAV). It may be undesirable for any vendor $v_j \neq v_i$ to gain unintended information about sensitive contents, even if, say, vendor v_j is trusted with control tasks for controlling the RTS. Similarly, the control laws from vendor v_j may contain a proprietary algorithm and vendor v_j may not want other vendors to gain knowledge about the algorithm. Protected communications and network monitoring/detection mechanisms are necessary but insufficient to deal with such threats. Therefore, *additional security tasks* may need to be added into the system to deal with such threats [25]. The security mechanisms could be protection, detection or response mechanisms, depending on the system requirements. For example, a sensor measurement correlation task may be added to detect sensor manipulation, a change detection task may be added to detect intrusions or additional state-cleansing tasks [26, 30, 27] can be added to deal with stealthy adversaries trying to glean sensitive information through side channels.

It is worth mentioning that the addition of such security mechanisms may necessitate changes to the schedule of real-time tasks. Contego is different from earlier work in which integration of security impacted the schedulability [27, 26, 30], required modification of the existing schedulers [39, 21], or necessitated architectural modifications [24, 42]. In contrast, Contego aims to integrate such security tasks *without* impacting the timeliness constraints (*i.e.,* schedulability) required for safe operation (in both modes) and retaining the original schedule of real-time tasks most of the time (*e.g.,* in PASSIVE mode when security tasks are executing opportunistically with lowest priority). We highlight that rather than designing specific intrusion detection tasks that target specific attack behaviors, the generic framework proposed in this work allows one to integrate a given security mechanism (referred to as

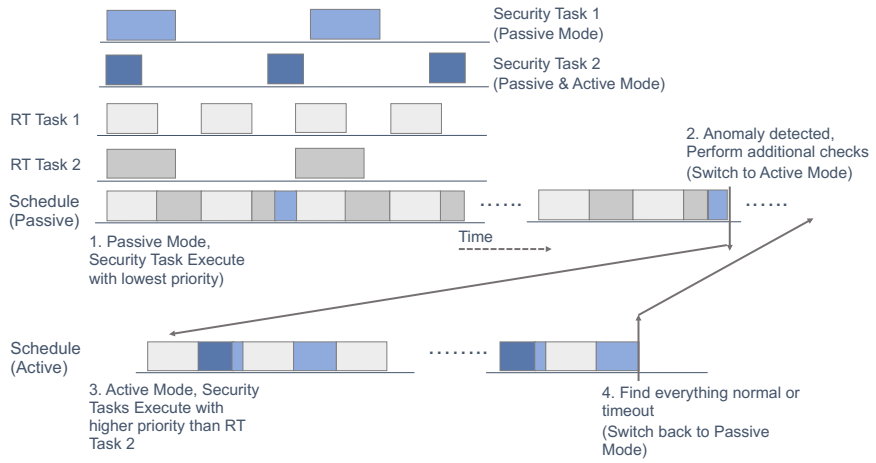


Figure 1 Contego: Flow of operations depicting the PASSIVE and ACTIVE modes for the security tasks.

security tasks) into the system without perturbing the system parameters (*e.g.,* period of the real-time tasks, execution order, *etc.*).

2.2 Overview of Contego

As illustrated in Fig. 1, Contego improves the security posture of the system (that contains a set of real-time tasks) by integrating additional security tasks and allowing them to execute in two different *modes* (*viz.,* PASSIVE and ACTIVE). If the system is deemed to be clean (*i.e.,* not compromised), security routines can execute *opportunistically*[2] (*e.g.,* when other real-time tasks are not running). However if any anomaly or unusual behavior is suspected, the security policy may switch to ACTIVE mode (*e.g.,* more fine-grained checking or response) and execute with *higher priority* for a *limited amount of time* (since our goal is to ensure security with minimum perturbation of the scheduling order of the real-time tasks). The security routines may go back to normal (*e.g.,* PASSIVE) mode if:

- No anomalous activity is found within a predefined time duration, say T^{AC}; or
- The intrusion is detected and malicious entities are removed (or an alarm triggered if human intervention is required).

Although we allow the security tasks to execute with higher priority than some of the real-time tasks in ACTIVE mode, the proposed framework ensures that the timeliness constraints (*e.g.,* deadlines) for *all* of the real-time tasks are always satisfied in *both* modes. By using this strategy, Contego not only enables *compatibility with legacy systems* (*e.g.,* in normal situation real-time scheduling order is not perturbed), but also provides *flexibility to promptly deal with anomalous behaviors* (*i.e.,* the security tasks are promoted to higher priority so that they can experience less preemption and achieve better response times).

[2] Which is also the default mode of operation.

2.3 System Model

2.3.1 Real-Time Tasks

In this paper we consider the widely used fixed-priority sporadic task model [28]. Let us consider a uniprocessor system consisting of m fixed-priority sporadic real-time tasks $\Gamma_R = \{\tau_1, \tau_2, \cdots, \tau_m\}$. Each real-time task $\tau_j \in \Gamma_R$ is characterized by (C_j, T_j, D_j), where C_j is the WCET, T_j is the minimum inter-arrival time (or period) between successive releases and D_j is the relative deadline. We assume that priorities are distinct and assigned according to the rate monotonic (RM) [22] order.

The processor utilization of τ_j is defined as $U_j = \frac{C_j}{T_j}$. Let $hp_R(\tau_j)$ and $lp_R(\tau_j)$ denote the sets of real-time tasks that have higher and lower priority than τ_j, respectively. We assume that the real-time task-set Γ_R is *schedulable* by a fixed-priority preemptive scheduling algorithm. Therefore, the worst-case response time w_i is less than or equal to the deadline D_i and the following inequality is satisfied for all tasks $\tau_j \in \Gamma_R$: $w_j \leq D_j$, where $w_j = w_j^{k+1} = w_j^k$ is obtained by the following recurrence relation [2]:

$$w_j^0 = C_j, \quad w_j^{k+1} = C_j + \sum_{\tau_h \in hp_R(\tau_j)} \left\lceil \frac{w_j^k}{T_h} \right\rceil C_h. \tag{1}$$

In Eq. (1), $\sum_{\tau_h \in hp_R(\tau_j)} \left\lceil \frac{w_j^k}{T_h} \right\rceil C_h$ is the worst-case interference to τ_j due to preemption by the tasks with higher priority than τ_j (*e.g.,* $hp_R(\tau_j)$). The recurrence will have a solution if $w_j^{k+1} = w_j^k$ for some k.

2.3.2 Security Tasks

With a view of integrating security into the system, let us add additional fixed-priority security tasks that will be executed in PASSIVE and ACTIVE modes. We model PASSIVE and ACTIVE mode security tasks as independent *sporadic tasks*. The PASSIVE and ACTIVE mode tasks are denoted by the sets $\Gamma_S^{pa} = \{\tau_1, \tau_2, \cdots, \tau_{n_p}\}$ and $\Gamma_S^{ac} = \{\tau_1, \tau_2, \cdots, \tau_{n_a}\}$, respectively. We assume that security tasks in both modes follow RM priority order. Each security task $\tau_i \in \{\Gamma_S^{pa} \cup \Gamma_S^{ac}\}$ is characterized by the tuple $(C_i, T_i^{des}, T_i^{max}, \omega_i)$, where C_i is the WCET, T_i^{des} is the most desired period between successive releases (hence $F_i^{des} = \frac{1}{T_i^{des}}$ is the desired execution frequency of a security routine) and T_i^{max} is the maximum allowable period beyond which security checking by τ_i may not be effective. The parameter $\omega_i > 0$ is a designer-provided weighting factor that may reflect the criticality of the security task[3] τ_i. Critical security tasks would have larger ω_i. The security tasks have implicit deadlines, *e.g.,* $D_i = T_i, \forall \tau_i$ that implies security tasks should complete before their next monitoring instance. We do not make any specific assumptions about the security tasks in different modes. For instance, both PASSIVE and ACTIVE mode task-sets may contain completely different sets of tasks (*e.g.,* $\{\Gamma_S^{pa} \cap \Gamma_S^{ac}\} = \emptyset$) or may contain (partially) identical tasks with different parameters (*e.g.,* period and/or criticality requirements).

In PASSIVE mode, security tasks are executed with *lower priority* than the real-time tasks. Hence the security tasks do not have any impact on real-time tasks and cannot perturb

[3] As an example, the default configuration of Tripwire [36], an intrusion detection system (IDS) for Linux that we use as case study in Section 6.2, has different criticality levels (*viz.,* weights), *i.e., High* (for scanning files that are significant points of vulnerability), *Medium* (for non-critical files that are of significant security impact) and so forth.

the real-time scheduling order. In ACTIVE mode, we allow the security tasks to execute with a priority higher than that of certain low priority real-time tasks. This provides us with a trade-off mechanism between security (*e.g.,* responsiveness) and system constraints (*e.g.,* scheduling order of real-time tasks). Since the task priorities are distinct, there are m priority-levels for real-time tasks (indexed from 0 to $m-1$ where level 0 is the highest priority). Among the m priority-levels, we assume that ACTIVE mode security tasks can execute with a priority-level up to l_S ($0 < l_S \leq m$), $l_S \in \mathbb{Z}$. Although any period T_i within the range $T_i^{des} \leq T_i \leq T_i^{max}$ is acceptable for PASSIVE (*e.g.,* $\tau_i \in \Gamma_S^{pa}$) and ACTIVE (*e.g.,* $\tau_i \in \Gamma_S^{ac}$) mode security tasks, the actual period T_i is not known a priori. Furthermore, for ACTIVE mode security tasks (*e.g.,* $\tau_i \in \Gamma_S^{ac}$), we need to find out the suitable priority level $l \in [l_S, m]$. Therefore our goal is to find the *suitable period* (for both PASSIVE and ACTIVE mode security tasks) as well as the *priority-level* (for ACTIVE mode security tasks) that achieve the best trade-off between schedulability and defense against security breaches without violating the real-time constraints.

3 Period Adaptation

As already mentioned, one fundamental problem in integrating security tasks is to determine *which* security tasks will be running *when*. This brings up the challenge of determining the *right periods* (*viz.,* the minimum inter-execution times) for the security tasks. For instance, some critical security routines 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. 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.

One may wonder why we cannot assign the desired period (*e.g.,* $T_i = T_i^{des}$) in both PASSIVE and ACTIVE modes and set the ACTIVE mode priority level as $l = l_S$ so that the security tasks can always execute with the desired frequency (*i.e.,* $F_i^{des} = \frac{1}{T_i^{des}}$) and experience less interference (*e.g.,* preemption) from real-time tasks. However, since our goal is to integrate security mechanisms in legacy systems with minimal[4] or no perturbation, setting $T_i = T_i^{des}$, $\forall \tau_i$ in either or both mode(s) may significantly perturb the real-time scheduling order. If the schedulability of the system is not analyzed after the perturbation, some (or all) of the real-time tasks may miss their deadlines and thus the main safety requirements of the system will be threatened. The same argument is also true for ACTIVE mode if we set $l = l_S$ (or arbitrarily from the range $[l_S, m]$) and do not perform schedulability analysis carefully.

3.1 Tightness of the Monitoring

As mentioned earlier, the actual period as well as the priority-levels of the security tasks are unknown and we need to *adapt* the periods within acceptable ranges. We measure the security of the system by means of *achievable periodic monitoring*. Let T_i be the period of the security task $\tau_i \in \{\Gamma_S^{pa} \cup \Gamma_S^{ac}\}$ that needs to be determined. Our goal is to minimize the

[4] In ACTIVE mode Contego does not introduce any timing violations for the real-time tasks, but their execution might be delayed due to interference from high-priority security tasks (*e.g.,* the tasks with priority-level $l \in [l_S, m]$).

gap between the achievable period T_i and the desired period T_i^{des} and therefore we define the following metric:

$$\eta_i = \frac{T_i^{des}}{T_i}, \tag{2}$$

that denotes the *tightness* of the frequency of periodic monitoring for the security task τ_i. Thus $\eta^{pa} = \sum_{\tau_i \in \Gamma_S^{pa}} \omega_i \eta_i$ and $\eta^{ac} = \sum_{\tau_i \in \Gamma_S^{ac}} \omega_i \eta_i$ denote the *cumulative tightness* of the achievable periodic monitoring for PASSIVE and ACTIVE mode, respectively. This monitoring frequency metric, provides for instance, one way to trade-off security with schedulability. Recall that if the interval between consecutive monitoring events is too large, the adversary may remain undetected and harm the system between two invocations of the security task. Again, a very frequent execution of security tasks may impact the schedulability of the real-time tasks. This metric $\eta^{(\cdot)}$ will allow us to execute the security routines with a frequency closer to the desired one while respecting the temporal constraints of the other real-time tasks.

3.2 Problem Overview

One may wonder why we cannot schedule the security tasks in the same way that the existing real-time tasks are scheduled. For instance, a simple approach to integrating security tasks in PASSIVE mode without perturbing real-time scheduling order is to execute security tasks at a *lower priority* than all real-time tasks. Hence, the security routines will be executing only during slack times when no other higher-priority real-time tasks are running. Likewise, in ACTIVE mode, security tasks can be executed at a lower priority than more critical, high-priority real-time tasks. Hence, the security tasks will only be executing when other real-time tasks with priority-levels higher than l_S are not running.

When both real-time and security tasks follow RM priority order, we can formulate a nonlinear optimization problem for PASSIVE mode with the following constraints that maximizes the cumulative tightness of the frequency of periodic monitoring:

(**P1**)

$$\max_{\mathbf{T^{pa}}} \quad \eta^{pa}$$

$$\text{Subject to:} \quad \sum_{\tau_i \in \Gamma_S^{pa}} \frac{C_i}{T_i} \leq (m+n_p)(2^{\frac{1}{m+n_p}} - 1) - \sum_{\tau_j \in \Gamma_R} \frac{C_j}{T_j} \tag{3a}$$

$$T_i \geq \max_{\tau_j \in \Gamma_R} T_j \qquad \forall \tau_i \in \Gamma_S^{pa} \tag{3b}$$

$$T_i^{des} \leq T_i \leq T_i^{max} \quad \forall \tau_i \in \Gamma_S^{pa} \tag{3c}$$

where $\mathbf{T^{pa}} = [T_1, T_2, \cdots, T_{n_p}]^\mathsf{T}$ is the optimization variable for PASSIVE mode that needs to be determined. The constraint in Eq. (3a) ensures that the utilization of the security tasks are within the remaining RM utilization bound [22]. The RM priority order for real-time and security tasks is ensured by the constraints in Eq. (3b), while Eq. (3c) ensures the restrictions on periodic monitoring.

Recall that in ACTIVE mode, we allow the security tasks to execute when the real-time tasks with priority-levels higher than l_S are not running. Hence, to ensure the RM priority order in ACTIVE mode, we need to modify the constraints in Eq. (3b) as follows:

$$T_i \geq \max_{\tau_j \in \Gamma_{R_{hp(l_S)}}} T_j, \quad \forall \tau_i \in \Gamma_S^{ac} \tag{4}$$

where $\Gamma_{R_{hp(l_S)}}$ represents the set of real-time tasks that are higher priority than level l_S. In addition, the constraints in Eq. (3a) and Eq. (3c) also need to be updated to consider ACTIVE mode task-sets (*e.g.,* Γ_S^{ac}) and the number of active mode security tasks (n_a). Thus for ACTIVE mode we can formulate an optimization problem similar to that of **P1** with the objective function: $\max_{\mathbf{T^{ac}}} \ \eta^{ac}$, where $\mathbf{T^{ac}} = [T_1, T_2, \cdots, T_{n_a}]^\mathsf{T}$ is the ACTIVE mode optimization variable.

One of the limitations of the above approach is that the overall system utilization is limited by the RM bound which has the theoretical upper bound of processor utilization only about $\lim_{n\to\infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \approx 69.31\%$ [22], where n is the total number of tasks under consideration. Further, the security tasks' periods need to satisfy the constraints in Eq. (3b) and Eq. (4) (for PASSIVE and ACTIVE modes, respectively) to follow RM priority order. In addition, instead of focusing only on optimizing the periods of the security tasks, Contego aims to provide a *unified* framework that can achieve other security aspects (*viz.,* responsiveness). Thus we follow an alternative approach similar to one we proposed in earlier work[5] [18]. Specifically, we had proposed to use a *server* [13] to execute security tasks. Our security server is motivated by the needs of hierarchical scheduling [35]. Under hierarchical scheduling, the system is composed of a set of components (*e.g.,* real-time tasks and a security server, in our context) and each of which comprises multiple tasks or subcomponents (*e.g.,* security tasks). The server abstraction not only allows us to provide better isolation between real-time and security tasks, but also enables us to integrate additional security properties (such as responsiveness) as we discuss in the following.

4 The Security Server

The server [13] is an abstraction that provides execution time to the security tasks according to a predefined scheduling algorithm. Our proposed security server is characterized by the *capacity* Q and *replenishment period* P. The server is executed with lowest-priority in PASSIVE mode. However, in ACTIVE mode, the server can switch to any allowable priority-level[6] within the range $[l_S, m]$.

4.1 Reformulation of the Period Adaptation Problem using Servers

When security tasks execute within the server, we need to modify the constraints in the period adaption problem considering the server parameters Q and P. In the following we briefly discuss how to customize the period adaptation problem with the inclusion of the server.

Let us use $UB_{\mathcal{S}(Q,P),\Gamma}$ to denote the utilization bound for the set of tasks Γ executing within the server. When the smallest period of the task is greater than or equal to $3P - 2Q$, it has been shown [31] that the upper bound of the utilization factor for the security tasks is given by $UB_{\mathcal{S}(Q,P),\Gamma} = n\left[\left(\frac{3-\frac{Q}{P}}{3-2\frac{Q}{P}}\right)^{\frac{1}{n}} - 1\right]$, where n is number of tasks in the set Γ.

Thus with the inclusion of the server in PASSIVE mode, we can modify the constraints in

[5] The approach we proposed in our earlier work [18] is analogous to the PASSIVE mode of Contego.
[6] Calculation of the server priority-level is described in Section 5.

Eqs. (3a) and (3b) as follows:

$$\sum_{\tau_i \in \Gamma_S^{pa}} \frac{C_i}{T_i} \leq n_p \left[\left(\frac{3 - \frac{Q^{pa}}{P^{pa}}}{3 - 2\frac{Q^{pa}}{P^{pa}}} \right)^{\frac{1}{n_p}} - 1 \right] \tag{5a}$$

$$T_i \geq 3P^{pa} - 2Q^{pa}, \quad \forall \tau_i \in \Gamma_S^{pa}. \tag{5b}$$

Therefore, selection of the periods for security tasks in PASSIVE mode is a nonlinear constrained optimization problem that can be formulated as follows:

(**P2**)

$$\max_{\mathbf{T^{pa}}} \sum_{\tau_i \in \Gamma_S^{pa}} \omega_i \frac{T_i^{des}}{T_i}, \quad \text{Subject to: (5a), (5b), (3c).}$$

where Q^{pa} and P^{pa} are the server capacity and replenishment period in PASSIVE mode, respectively. The formulation of the PASSIVE mode period adaptation problem presented above is similar to that we proposed in earlier work [18]. Similarly, in ACTIVE mode, the period adaptation problem can be reformulated as follows:

(**P3**)

$$\max_{\mathbf{T^{ac}}} \sum_{\tau_i \in \Gamma_S^{ac}} \omega_i \frac{T_i^{des}}{T_i}$$

$$\text{Subject to: } \sum_{\tau_i \in \Gamma_S^{ac}} \frac{C_i}{T_i} \leq n_a \left[\left(\frac{3 - \frac{Q^{ac}}{P^{ac}}}{3 - 2\frac{Q^{ac}}{P^{ac}}} \right)^{\frac{1}{n_a}} - 1 \right] \tag{7a}$$

$$T_i \geq 3P^{ac} - 2Q^{ac} \quad \forall \tau_i \in \Gamma_S^{ac} \tag{7b}$$

$$T_i^{des} \leq T_i \leq T_i^{max} \quad \forall \tau_i \in \Gamma_S^{ac} \tag{7c}$$

where Q^{ac} and P^{ac} are the server capacity and replenishment period in ACTIVE mode, respectively.

4.2 Selection of the Server Parameters

The period adaptation problem illustrated in Section 4.1 is derived based on a given set of server parameters, *e.g.,* $(Q^{(\cdot)}, P^{(\cdot)})$. However, a fundamental problem is to find a suitable pair of server capacity $Q^{(\cdot)}$ and replenishment period $P^{(\cdot)}$ that respects the real-time constraints of the tasks in the system. Our approach to selecting the server parameters in PASSIVE and ACTIVE mode is described below.

4.2.1 Parameter Selection in Passive Mode

Recall that in PASSIVE mode, the server will execute with the lowest priority to have compatibility with existing real-time tasks. Since the security tasks execute within the server, we need to ensure the following two constraints:

- *The server is schedulable*: that is the server's capacity and interference from higher priority real-time tasks are less than the replenishment period; and
- *The security tasks are schedulable*: the minimum *supply* by the server to the security tasks is greater than the worst-case workload generated by the security tasks.

Note that since the server is running with lowest priority, the real-time constraints (*e.g.,* $w_j \leq D_j, \forall \tau_j \in \Gamma_R$) and the task execution order are not affected in the PASSIVE mode. Based on the above two constraints, we illustrate an approach for determining the server parameters by formulating it as a *constraint optimization problem.*

The security server is referred to as *schedulable* if the worst-case response time of the server does not exceed its replenishment period [13]. Thus, following an approach similar to ones in earlier work [18, 40], the *server schedulability constraint* can be represented as follows:

$$Q^{pa} + \Delta_{\mathcal{S}^{pa}} \leq P^{pa} \tag{8}$$

where $\Delta_{\mathcal{S}^{pa}} = \sum_{\tau_h \in hp_R(\tau_{\mathcal{S}^{pa}})} \left(\frac{P^{pa}}{T_h} + 1\right) C_h$ is the worst-case interference experienced by the server when preempted by the higher priority real-time tasks. In the above equation, the set of real-time tasks with higher priority than the server (*i.e.,* $hp_R(\tau_{\mathcal{S}}^{pa}) = \Gamma_R$) is fixed.

Let us use $hp_S^{pa}(\tau_i)$ to denote the set of PASSIVE mode security tasks that are higher priority than $\tau_i \in \Gamma_S^{pa}$. To ensure schedulability of the security tasks, we can derive the *minimum supply* of the server delivered to the security tasks by using the periodic resource model from the literature [35, 40, 18]. In particular, the constraints on the server supply to ensure *schedulability of the security tasks* [18] can be expressed as:

$$\frac{Q^{pa}}{P^{pa}} \left[T_i - (P^{pa} - Q^{pa}) - \Delta_{\mathcal{S}^{pa}}\right] \geq I_i^{pa}, \quad \forall \tau_i \in \Gamma_S^{pa} \tag{9}$$

where $I_i^{pa} = C_i + \sum_{\tau_h \in hp_S^{pa}(\tau_i)} \left\lceil \frac{T_i}{T_h} \right\rceil C_h$ is the worst-case workload generated by the security task τ_i and $hp_S^{pa}(\tau_i)$ during the time interval of T_i. This workload is a constant for a given input.

Since we need to ensure maximal processor utilization for the security tasks without violating the real-time constraints of the system, we define the following objective function: $\max_{Q^{pa}, P^{pa}} \frac{Q^{pa}}{P^{pa}}$. With this objective function and the constraints in Eqs. (8)–(9), the PASSIVE mode server parameter selection problem can be formulated as follows:

(**P4**)

$$\max_{Q^{pa}, P^{pa}} \frac{Q^{pa}}{P^{pa}}, \quad \text{Subject to: (8), (9)}$$

where server parameters Q^{pa} and P^{pa} are the optimization variables.

4.2.2 Parameter Selection in Active Mode

In ACTIVE mode, the security server is *no longer the lowest priority task.* Since the server can execute with priority l_S, there could be up to $m - l_S$ low priority real-time tasks than that of the server. Thus we need to ensure the schedulability of the real-time tasks that are executing with a priority lower than the server. Hence, in addition to the constraints described in Section 4.2.1 (*i.e.,* Eqs. (8)-(9)), we need to consider the following:

- *The real-time tasks with lower priority than the server are schedulable*: that is, the interferences from the server and other higher priority real-time tasks do not violate the deadlines for these low-priority tasks.

We therefore define the following constraints to ensure the *schedulability of the low-priority real-time tasks*:

$$C_j + \sum_{\tau_h \in hp_R(\tau_j)} \left\lceil \frac{D_j}{T_h} \right\rceil C_h + \left(\frac{D_j}{P^{ac}} + 1 \right) Q^{ac} \leq D_j, \quad \forall \tau_j \in lp_R(\tau_{\mathcal{S}}^{ac}) \tag{11}$$

where $\sum_{\tau_h \in hp_R(\tau_j)} \left\lceil \frac{D_j}{T_h} \right\rceil C_h$ is the interference experienced by τ_j from other real-time tasks and $\left(\frac{D_j}{P^{ac}} + 1 \right) Q^{ac}$ is the worst-case interference caused to τ_j by the server in ACTIVE mode. As illustrated in Section 5, we iterate through the allowable priority ranges (*e.g.,* $[l_S, m]$) to find the server priority in ACTIVE mode. Note that for a given priority-level, the set of tasks $lp(\tau_{\mathcal{S}}^{ac})$ is predefined. Thus the only variables for the constraints in Eq. (11) are the server capacity Q^{ac} and replenishment period P^{ac}.

Let us use $hp_S^{ac}(\tau_i)$ to denote the set of ACTIVE mode security tasks that are higher priority than $\tau_i \in \Gamma_S^{ac}$. Just as in **P4** we can now formulate the ACTIVE mode parameter selection problem as follows:

(**P5**)

$$\max_{Q^{ac}, P^{ac}} \frac{Q^{ac}}{P^{ac}}, \text{ Subject to: (11) and}$$

$$Q^{ac} + \sum_{\tau_h \in hp_R(\tau_{\mathcal{S}^{ac}})} \left(\frac{P^{ac}}{T_h} + 1 \right) C_h \leq P^{ac} \tag{12a}$$

$$\frac{Q^{ac}}{P^{ac}} \left[T_i - (P^{ac} - Q^{ac}) - \Delta_{\mathcal{S}^{ac}} \right] \geq I_i^{ac} \quad \forall \tau_i \in \Gamma_S^{ac} \tag{12b}$$

where the set of real-time tasks with higher priority than the server (*i.e.,* $hp_R(\tau_{\mathcal{S}}^{ac}) \subset \Gamma_R$) is a constant for a given priority-level and $I_i^{ac} = C_i + \sum_{\tau_h \in hp_S^{ac}(\tau_i)} \left\lceil \frac{T_i}{T_h} \right\rceil C_h$ is the worst-case workload generated by the security task τ_i and $hp_S^{ac}(\tau_i)$. Note that the schedulability of the higher priority real-time tasks (*e.g.,* $\forall \tau_j \in hp_R(\tau_{\mathcal{S}}^{ac})$) is already ensured by definition.

▶ Remark. The formulation of the period adaptation and server parameter selection problems are nonlinear constraint optimization problems and are nontrivial to solve in their current form. However, these problems can be transformed into a geometric programming (GP) [6] problem. In addition, it is also possible to reformulate the non-convex GP representation into equivalent convex form that can be solved using known algorithms such as *interior point* [7, Ch. 11] method. For details of this reformulation, we refer the readers to earlier work [18].

4.3 Discussion on Mode Switching

As mentioned earlier, by default, Contego operates in PASSIVE mode. However, when a malicious activity is suspected, a PASSIVE-to-ACTIVE mode change request will be issued. Similarly, an ACTIVE-to-PASSIVE mode change request will be placed if the system seems clean after fine-grained checking, or a malicious entity is found and removed. In steady-state (*e.g.,* when security tasks are executing in PASSIVE or ACTIVE mode), the schedulability of the real-time tasks is already guaranteed by the analysis presented in Section 4.2.

When Contego switches from PASSIVE mode to ACTIVE mode, the schedulability of real-time tasks will not be affected. The reason this that *all* the real-time tasks are higher priority than the security tasks in PASSIVE mode and hence do not suffer any additional

interference from security tasks during mode change. Therefore, the schedulability of real-time tasks during PASSIVE-to-ACTIVE mode switching is already covered by steady-state analysis (Section 4.2.1).

During ACTIVE-to-PASSIVE mode switching, observe that schedulability of the real-time tasks that have a priority higher than the server (*i.e.,* $hp_R(\tau_S^{ac})$) is not affected. When the mode switch request is issued, the ACTIVE mode server (and the security tasks) stop execution and the control is then switched to the lowest priority PASSIVE mode server. Note that the constraints in Eq. (11) that ensures the schedulability of the low-priority real-time tasks already captures the worst-case interference introduced by the server. Hence the server will not impose any more interference (even if the mode switch is performed in the middle of the execution of a busy interval) on the low-priority real-time tasks than what we have calculated in the steady-state analysis (Section 4.2.2). Therefore if both the PASSIVE and ACTIVE modes task-sets are schedulable, the system will also be schedulable with mode changes.

5 Algorithm Development

We develop a simple scheme to obtain the security task's period (for both PASSIVE and ACTIVE mode) and priority-level (for ACTIVE mode). The overall algorithm, Algorithm 1, works as follows.

To find the PASSIVE mode parameters, we initialize the security task's period with the desired period and solve the server parameter selection problem **P4** (Lines 10–11). If there exists a solution (*e.g.,* the constraints are satisfied), we then obtain the periods of the security tasks by solving **P2** (Line 13). In the event that neither of these optimization problems returns a solution, we report the task-set as unschedulable (Line 20), since it is not possible to execute security tasks opportunistically without violating real-time constraints.

To select ACTIVE mode parameters, the algorithm iterates through each of the acceptable priority-levels $[l_S, m]$ and tries to obtain the periods that maximize tightness for periodic monitoring without violating the real-time constraints (Lines 26–36). If there exists a solution (*e.g.,* constraints in **P5** and **P3** are mutually consistent), we store the solution in a candidate list. The algorithm then finds the best priority-level from the candidate solution sets that provides the maximum tightness (Line 39). In the event that no candidate solutions are found for any of the allowable priority ranges, the algorithm reports the task-set as unschedulable.

If *both* the PASSIVE and ACTIVE mode tasks are schedulable, then Algorithm 1 returns the corresponding periods and the ACTIVE mode priority-level (Line 4). Otherwise, the system is considered as unschedulable (Line 7) since it is not possible to integrate security tasks with desired requirements. This unschedulability result hints that the designers of the system should update system parameters (*e.g.,* the number of security tasks, desired and maximum allowable periods of the security tasks, periods of the real-time tasks, if permissible, *etc.*) in order to integrate security mechanisms.

6 Evaluation

We evaluate Contego with randomly generated synthetic workloads (Section 6.1) as well as a proof-of-concept implementation on an ARM-based embedded development board and real-time Linux (Section 6.2).

Algorithm 1 Feasibility Checking and Parameter Selection

Input: Set of real-time tasks, Γ_R, PASSIVE and ACTIVE mode security tasks Γ_S^{pa} and Γ_S^{ac}, allowable priority ranges $[l_S, m]$

Output: The tuple $\{l^*, \mathbf{T^{pa}}, Q^{pa}, P^{pa}, \mathbf{T^{ac}}, Q^{ac}, P^{ac}\}$, *e.g.,* ACTIVE mode server priority-level, ACTIVE and PASSIVE mode periods of the security tasks and ACTIVE and PASSIVE mode server parameters if the task-set is schedulable; Unschedulable otherwise

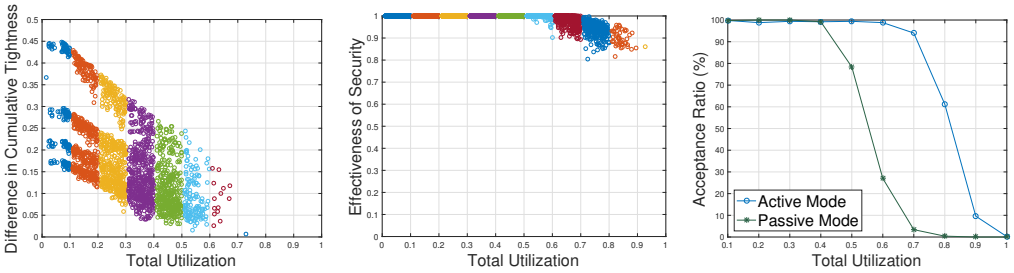
Obtain PASSIVE and ACTIVE mode parameters using the functions PassiveModeParamSelection(Γ_R, Γ_S^{pa}) and ActiveModeParamSelection(Γ_R, Γ_S^{ac}, l_S)
if Solution Found in BOTH Modes **then**
  / return the parameters */*
  return $\{l^*, \mathbf{T^{pa}}, Q^{pa}, P^{pa}, \mathbf{T^{ac}}, Q^{ac}, P^{ac}\}$
else
  / not possible to integrate security tasks in the system */*
  return Unschedulable
end if

---

function PassiveModeParamSelection(Γ_R, Γ_S^{pa})
  Initialize PASSIVE mode period $T_i := T_i^{des}, \forall \tau_i \in \Gamma_S^{pa}$
  Solve **P4** to obtain server parameters
  if SolutionFound **then**
    Solve **P2** to obtain security periods
    if SolutionFound **then**
      / return the parameters */*
      return $\mathbf{T^{pa}}$, Q^{pa}, P^{pa} where Q^{pa}, P^{pa} and $\mathbf{T^{pa}}$ are the solutions obtained by **P4** and **P2**
    end if
  else
    / unable to integrate* PASSIVE *mode security tasks */*
    return Unschedulable
  end if
end function

---

function ActiveModeParamSelection(Γ_R, Γ_S^{ac}, l_S)
  Schedulable := **false**
  Initialize ACTIVE mode security task's period $\mathbf{T}(l')_{\forall l' \in [l_S, m]} := [T_i^{des}]^{\mathsf{T}}_{\forall \tau_i \in \Gamma_S^{ac}}$
  for each priority level $l' \in [l_S, m]$ **do**
    Solve **P5** to obtain server parameters
    if SolutionFound **then**
      Solve **P3** to obtain security periods
      if SolutionFound **then**
        / store the parameters for priority level l' where Q^*, P^* and $\mathbf{T}^*$ are the solutions obtained by* **P5** *and* **P3** **/*
        $Q(l') := Q^*, P(l') := P^*, \mathbf{T}(l') := \mathbf{T}^*$
        Schedulable := **true**
      end if
    end if
  end for
  / obtain the parameters that provide best metric */*
  if Schedulable **then**
    Find the priority-level l^* from the solution vector $\mathbf{T}(l')_{\forall l' \in [l_S, m] |}$ tasks at l' is schedulable that gives the maximum cumulative tightness $\eta^{ac} = \sum_{\tau_i \in \Gamma_S^{ac}} \eta_i$
    Set $\mathbf{T^{ac}} := \mathbf{T}(l^*)$, $Q^{ac} := Q(l^*)$, $P^{ac} := P(l^*)$
    / return the parameters */*
    return l^*, $\mathbf{T^{ac}}$, Q^{ac}, P^{ac}
  else
    / unable to integrate* ACTIVE *mode security tasks */*
    return Unschedulable
  end if
end function



(a) Difference in cumulative tightness.

(b) Effectiveness of security in different system utilization.

(c) Schedulability of task-sets in both modes.

Figure 2 Experiments with synthetic task-sets: **(a)** PASSIVE mode vs. ACTIVE mode: difference in cumulative tightness of achievable periodic monitoring, $\eta^{av} - \eta^{pa}$. Non-zero difference indicates that the ACTIVE mode tasks achieve better tightness than PASSIVE mode tasks. Each of the data points represents schedulable task-sets. **(b)** The effectiveness of security vs. total utilization of the system. The closer the y-axis values to 1, the nearer each security task's period is to the desired period. **(c)** Schedulability of real-time and security tasks in both modes. The acceptance ratio is defined by the ratio of the number of accepted task sets over the total number of generated tasks. For each of the data points, 500 individual task-sets were tested. In figure (a) and (b), task-sets from different base-utilization groups are distinguished by different colors.

6.1 Experiment with Synthetic Task-sets

6.1.1 Simulation Setup

In order to generate task-sets with an even distribution of tasks, we grouped the real-time and security task-sets by base-utilization from $[0.01 + 0.1 \cdot i, 0.1 + 0.1 \cdot i]$, where $i \in \mathbb{Z} \wedge 0 \leq i \leq 9$. Each utilization group contained 500 task-sets. In other words, a total of 5000 task-sets were tested for each of the experiments. The utilization of the real-time and security tasks were generated by the UUniFast [4] algorithm and we used GGPLAB [29] to solve the optimization problems.

We used the parameters similar to those used in earlier research [26, 18]. In particular, each task-set instance contained $[3, 10]$ real-time and $[2, 5]$ security tasks in each of the modes. Each real-time task $\tau_j \in \Gamma_R$ had a period $T_j \in [10 \text{ ms}, 100 \text{ ms}]$ and we assumed $l_S = \lceil 0.4m \rceil$. The desired periods for the security tasks $\forall \tau_i \in \{\Gamma_S^{pa} \cup \Gamma_S^{ac}\}$ were selected from $[1000 \text{ ms}, 3000 \text{ ms}]$ and the maximum allowable period was assumed to be $T_i^{max} = 10T_i^{des}$. We considered $\omega_i = 1$, $\forall \tau_i \in \{\Gamma_S^{pa} \cup \Gamma_S^{ac}\}$ and the total utilization of the security tasks was assumed to be no more than 30% of the real-time tasks.

6.1.2 Results

6.1.2.1 Impact on Cumulative Tightness

In Fig. 2a one can see the difference in the tightnesses of the periodic monitoring obtained by PASSIVE and ACTIVE mode (*i.e.*, $\eta^{ac} - \eta^{pa}$). For fair comparison we used the same task-sets for both modes. The x-axis of Fig. 2a represents the total system utilization (*e.g.*, utilization of both real-time and security tasks). The positive values in the y-axis of Fig. 2a imply that the ACTIVE mode tasks obtain better tightness that the PASSIVE mode tasks.

The figure shows that ACTIVE mode tasks can achieve better cumulative tightness, and that the cumulative tightness η^{pa} is comparatively better in low to medium utilization. The main reason is that in ACTIVE mode security tasks are allowed to execute with higher priority,

that causes less interference and eventually increases the feasible region in the optimization problems (and hence provides better tightness). For higher utilizations the difference is close to zero. This is because, as utilization increases there is less slack in the system, making it difficult to schedule security tasks frequently and resulting in similar levels of tightness for both modes.

6.1.2.2 Effectiveness of Security

The parameter $\eta^{(\cdot)}$ is given by the total number of security tasks and provides insights on cumulative measures of security. However, in this experiment (refer to Fig. 2b) we wanted to measure the effectiveness of the security of the system by observing whether *each* of the security tasks in any mode can achieve an execution frequency closer to the desired one. Hence we used the following metric: $\xi = 1 - \frac{\|\mathbf{T}^* - \mathbf{T^{des}}\|_2}{\|\mathbf{T^{max}} - \mathbf{T^{des}}\|_2}$ where $\mathbf{T}^*$ is the solution obtained from Algorithm 1, $\mathbf{T^{des}} = [T_i^{des}]_{\forall \tau_i}^{\mathsf{T}}$ and $\mathbf{T^{max}} = [T_i^{max}]_{\forall \tau_i}^{\mathsf{T}}$ are the desired and maximum period vector (refer to Section 6.1.1), respectively, and $\|\cdot\|_2$ denotes the Euclidean norm. The closer the value of ξ to 1, the nearer each of the security task's period is to the desired period. As the total utilization increases, the feasible set of the period adaptation problem that respects all constraints in the optimization problems becomes more restrictive. As a result, we see the degradation in effectiveness (in terms of ξ) for the task-sets with higher utilization. However, from our experiments we find that Contego can achieve periods that are *within* 18% *of the desired periods.*

6.1.2.3 Impact on the Schedulability

We used the *acceptance ratio* metric to evaluate schedulability. The acceptance ratio (y-axis in Fig. 2c) is defined as the number of accepted task-sets (*e.g.,* the task-sets that satisfied all the constraints) over the total number of generated ones. As depicted in Fig. 2c the ACTIVE mode task-set achieves better schedulability compared to the PASSIVE ones. Recall that ACTIVE mode task-sets can be promoted up to priority level l_S. As a result ACTIVE mode security tasks potentially experience less interference than the PASSIVE ones. This flexibility gives the optimization routines a larger feasibility region to satisfy all the constraints.

6.2 Experiment with Security Applications in an Embedded Platform

To observe the performance of the proposed scheme in a practical setup, we implemented Contego on an embedded platform. Our experimental platform [3] was configured with 1 GHz ARM Cortex-A8 single-core processor and 512 MB RAM. We used Linux as the operating system – that allowed us to utilize the existing Linux-based IDSes (refer to Section 6.2.2) for the evaluation. Since the vanilla Linux kernel is unsuitable for hard real-time scheduling, we enabled the real-time capabilities with the Xenomai [38] 2.6.3 real-time patch (kernel version 3.8.13-r72) on top of an embedded Debian Linux console image.

We measured the WCET of the real-time and security tasks using ARM cycle counter registers (*e.g.,* CCNT), giving us nanosecond-level precision. Since these registers are not enabled by default, we developed a Linux kernel module to access the registers from our application codes. Our prototype implementation was developed in C and uses a fixed-priority scheduler powered by the Xenomai real-time patch. Sporadic real-time and security tasks in the system were defined by Xenomai `rt_task_create`() function and were suspended after the completion of corresponding instances using the `rt_task_wait_period`() function.

Table 1 Real-time task parameters for the UAV control system.

Task	Function	Period (ms)
Guidance	Select the reference trajectory (*i.e.,* altitude and heading)	1000
Controller	Execute closed-loop control functions (*e.g.,* actuator commands)	5000
Reconnaissance	Read radar/camera data, collect sensitive information and send data to the base control station	10000

Table 2 Security tasks used in the experiments.

Task	Function	Mode
Check own binary of the security routine (Tripwire)	Scan files (*viz.,* compare their hash value) in the following locations: `/usr/sbin/siggen`, `/usr/sbin/tripwire`, `/usr/sbin/twadmin`, `/usr/sbin/twprint`, `/usr/local/bro/bin`	ACTIVE
Check critical executables (Tripwire)	Scan file-system binary (`/bin`, `/sbin`)	ACTIVE and PASSIVE
Check critical libraries (Tripwire)	Scan file-system library (`/lib`)	ACTIVE
Monitor network traffic (Bro)	Scan predefined network interface (`en0`)	ACTIVE and PASSIVE

6.2.1 Real-time Tasks

For a real-time application, we considered a UAV control system (refer to Table 1). We implemented it using an open-source UAV model [37]. The original application codes were based on the STM32F4 micro-controller (ARM Cortex M4) and developed for FreeRTOS [16]. Because of differences in library support and execution semantics, we updated the source codes accordingly and ported them to Linux.

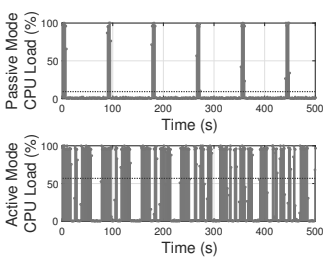
6.2.2 Security Tasks

To integrate security in the aforesaid control system, we included additional security tasks. For the security tasks, we considered two lightweight open-source intrusion detection mechanisms, (i) Tripwire [36], that detects integrity violations by storing clean system state during initialization and using it later to detect intrusions by comparing the current system state against the stored clean values, and (ii) Bro [8] that monitors anomalies in network traffic. As Table 2 shows, we consider several security tasks in both modes, *e.g., protecting security task's own binary files*, *protecting system binary and library files*, *monitoring network traffic*. In each mode, we set the desired and maximum allowable periods of the security tasks such that utilization of the security tasks did not exceed 50% of the total system utilization.

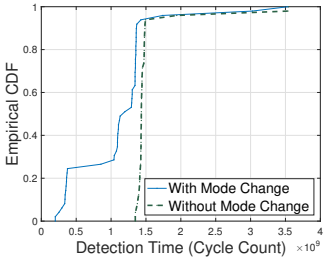
6.2.3 Experience and Evaluation

6.2.3.1 Performance Impact in Different Modes

In the first set of experiments, we measured the average CPU load when the security tasks were executing in PASSIVE and ACTIVE modes. For that, we executed the security tasks



(a) CPU load in different modes.



(b) Empirical CDF of detection time.

Figure 3 Experiments with synthetic task-sets: **(a)** The CPU load when the security tasks executed in PASSIVE (top) and ACTIVE (bottom) mode, respectively. The dotted line represents average load over the observation duration (500 s). **(b)** The empirical distribution of time to detect the intrusions when mode change was allowed vs when security tasks were run only in PASSIVE mode. We used ARM cycle counter registers to measure the detection time. A total of 50 individual experiment instances were examined to obtain the timing traces.

independently for 500 s in PASSIVE and ACTIVE modes and observed the CPU load using `/proc/stat` interface (represents the y-axis of Fig. 3a).

As Fig. 3a shows, running security tasks in ACTIVE mode increased the average CPU load compared to running them in PASSIVE mode. This is because ACTIVE mode contains more security tasks (*e.g.,* 4 compared to 2, refer to Table 2) and they execute more frequently than in PASSIVE mode. Because of the nature of applications, most RTS prefer predictability over performance. The overhead of running security tasks in ACTIVE mode comes with increased security guarantees that will suffice for many RTS.

6.2.3.2 Impact on Detection Time

To study the detection performance we injected malicious code into the system that mimics anomalous behaviors. We assumed that an attacker can take over[7] one of the low-priority real-time tasks (referred to as the victim task) and is able to insert malicious code that can execute with a privilege similar to that of legitimate tasks. We launched the attack at both the *network* and *host*-level. We defined network-level DoS attacks as too many rejected usernames and passwords submitted from a single address and used a real FTP DoS trace [17] to demonstrate the attack. Malware (such as LRK, tOrn, Adore, *etc.*) in general-purpose Linux environments causes damage to the system by modifying or overwriting the system binary [14, Ch. 5]. Thus we follow a similar approach to demonstrate a host-level attack, *viz.,* we injected ARM shellcode [33] to override the victim task's code and launched the attack by modifying the contents in the file-system binary. We obtained the periods of the security tasks in both modes by solving the period adaptation problem (Algorithm 1) and set it as the period of security tasks (by using the Xenomai `rt_task_set_periodic`() function). For each of the experiments, the work-flow was as follows. We started with a clean (*e.g.,* uncompromised) system state, launched the DoS attack at any random time of the program execution and then injected the shellcode after a random interval, and finally logged the time required by security tasks to detect the attacks. Initially the security tasks ran in PASSIVE

[7] One way to override a task could be to use an approach similar to one presented in the literature [11] that exploits the deterministic behavior of the real-time scheduling.

mode. When the network-level attack was suspected by the security task (*e.g.,* Bro), a mode change request was placed and the control was switched to ACTIVE mode with the corresponding ACTIVE mode tasks (see Table 2). As mentioned in Section 2.2, our focus is *not* on the effectiveness of a particular IDS here but on the effectiveness of integration of the IDSes into RTS. Therefore we controlled the experimental environment so that the results were not affected by the false positive/negative rates of the IDS used in the evaluation. In particular, both of the launched attacks were detectable by the respective IDSes used in the evaluation. Detection times were measured using ARM cycle counter registers (CCNT). To ensure the accuracy of the detection time measurements, we disabled all the frequency scaling features in the kernel (by using the `cpufrequtils` utility) and allowed the platform to execute with a constant frequency (*e.g.,* 1 GHz, the maximum frequency of our experimental platform).

We compared the performance of Contego with that of an earlier approach [18] that has no provision for mode changes and in which the security tasks are run with the lowest priority (similar to the PASSIVE mode of operation in Contego). Specifically, we measured the time to detect both the host and network-level intrusions, and plot the empirical cumulative distribution function (CDF) of those detection times in Fig. 3b. The x-axis in Fig. 3b represents the detection time (in cycle count) and the y-axis represents the probability that the attack would be detected by that 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 the time taken to detect the attack in the i-th experimental observation, and $\jmath$ represents the x-axis values (*viz.,* the detection times in cycle count) in Fig. 3b. The indicator function $\mathbb{I}_{[\cdot]}$ outputs 1 if the condition $[\cdot]$ is satisfied and 0 otherwise.

From Fig. 3b we can see that Contego provides better detection time (*i.e.,* fewer cycle counts required to detect the intrusions). From our experiments we find that *on average* Contego detects attacks 27.29% faster than the reference scheme does. The approach from the literature [18] allows the security tasks to run only when other real-time tasks are not running, leading to more interference (*e.g.,* higher response times), and does not provide any mechanisms to adapt against abnormal behaviors (*e.g.,* the DoS attack in the experiments). In contrast, Contego allows quick response to anomalies (by switching to ACTIVE mode when a DoS attack is suspected). Since ACTIVE security tasks can run with higher priority and less interference without impacting the timeliness constraints of real-time tasks, Contego had a superior detection rate in general for most of the experiments without impacting safety.

7 Discussion

Although Contego provides an integrated approach to guarantee safety and security in RTS, this framework can be extended in several directions. In the following, we briefly analyze Contego against different threat models and discuss the limitations of the current framework with possible directions of improvement.

7.1 Threat Analysis

The security mechanism will collapse if the adversary can compromise *all* the security tasks. To do so, the adversary would need to intrude into the system, remain undetected and monitor the schedule [11] (to override the security tasks) *over a long period of time*. Guaranteeing the integrity of the security tasks is an interesting research problem by itself and will be investigated in our future work. While compromising all the security tasks

could be *difficult* in practice, it nevertheless would be worthwhile to harden the security posture of Contego further by *randomizing task schedules* while guaranteeing the safety of the real-time tasks by using approaches similar to one recently proposed in the literature [41]. Randomizing the schedule of real-time and security tasks reduces the determinism (and thus the predictability of security tasks' execution) and further reduce the chance of information leakage. Randomizing task schedules in RTS, unlike traditional systems, is not straightforward since it leads to priority inversions [32] that, in turn, cause missed deadlines, and hence, put the safety of the system at risk. We intend to incorporate randomization protocols on top of Contego in future work.

The underlying detection algorithms in security tasks could raise false positive errors that may cause the system to switch modes unnecessarily. Again, a clever adversary may remain undetected and provide a fake indication of malicious activity. This may cause Contego to frequently switch modes thus reducing performance and availability. Although Contego *guarantees that the system will remain schedulable* (and hence safe) even with mode changes (refer to Section 4.3), running of security tasks in ACTIVE mode could impose additional overheads (*i.e.,* increased load as we have seen in Fig. 3a) that designers of the system may want to avoid. The false positive/negative errors can be mitigated by carefully designing the detection algorithms based on application requirements. Further, we argue that forced mode changes would require an adversary to intrude in the system and *remain undetected for a long time*. In practice that could be *difficult* and *unlikely* in the presence of several intrusion detection tasks.

7.2 Limitations and Improvement

In Contego each security task has a desired frequency of execution for better security coverage. Security tasks so far have been treated as *independent* and *preemptive*, but in practice, some security monitoring may need *atomicity* or non-preemptive execution. Further, security tasks may have *dependencies* wherein one task depends on the output from one or more other tasks. For example, an anomaly detection task might depend on the outputs of multiple scanning tasks, or, the scheduling framework might need to follow certain *precedence constraints* for security tasks. In order to ensure the integrity of monitoring security, the security application's own binary might need to be examined first before it checks the system binary files. In that case, the *c*umulative tightness of the achievable periodic monitoring proposed in Section 3 might no longer be a reasonable metric. Constraints to ensure that the dependent security tasks are executed often enough should be included and the optimization problem may need to be reformulated and evaluated with different metrics.

While *time-to-detect* is a useful metric, it is hard to quantify in a comprehensive way as it depends on a number of factors such as the efficacy of monitoring tasks, the kind of intrusion *etc.* and is a lagging metric. Identifying and designing better security metrics is an important and challenging problem. In future work we will undertake it in the narrow context of integrating monitoring and detection tasks into RTS.

8 Related Work

In our earlier work [18] we proposed to use a server to integrate security tasks and execute them opportunistically at a lower priority than real-time tasks. That approach was useful for legacy RTS where perturbing the schedule of real-time tasks was not an option – however, the downside was longer time for detection. In contrast, Contego can respond to anomalous

activities in an adaptive manner and provide improved monitoring frequency and detection time when needed.

A state cleanup mechanism has been introduced [26], and further generalized [30, 27] such that the fixed-priority scheduling algorithm was modified to mitigate information leakage through shared resources. A new scheduler [39] and enhancements to an existing dynamic priority scheduler [21] were proposed to meet real-time requirements while maximizing the level of security achieved. Researchers have also proposed a schedule obfuscation method [41] aimed at randomizing the task schedule while providing the necessary real-time guarantees. Such randomization techniques can improve the security posture by minimizing the predictability of the deterministic RTS scheduler. Recent work [24, 42] on architectural frameworks has aimed to protect RTS against security threats. However, those approaches came at the cost of reduced schedulability or may require architectural/scheduler-level modifications. In comparison, Contego aims to integrate security *without* any significant modification of the system properties and does *not* violate the temporal constraints or schedulability of the real-time tasks.

Although not in the context of security in RTS, there exists other work [5] in which the authors statically assign the periods for multiple independent control tasks by considering control delay as a cost metric and estimating the delay through an approximate response time analysis. In contrast, our goal is to ensure security without violating the timing constraints of the real-time tasks. Hence, instead of minimizing response time, we attempt to assign the best possible periods and priority-levels so that we can minimize the perturbation between the achievable period and desired period for all the security tasks.

An on-demand fault detection and recovery mechanism has been proposed [23] in which the system can operate in different modes. Specifically, when a fault is detected, a high-assurance controller is activated to replace the faulty high-performance controller. While fault-tolerance may also be a design consideration, Contego focuses primarily on integrating mechanisms that can foil cyber-attacks. There also exist work in the context of mixed-criticality systems (MCS) where application tasks of different criticality requirements (*e.g.,* deadline and execution time) share same computation and/or communication resources (refer to literature [9] for a survey of MCS). MCS is different than the problem considered in this work due to the fact that security properties (*i.e.,* adaptive switching depending on runtime behavior or frequent execution of monitoring events for faster detection) are often different than temporal requirements (*e.g.,* satisfying deadline constraints for mixed-criticality tasks). However, the theory and concepts emerged from MCS can also be applied to the real-time security problems to further harden the security posture of future RTS.

9 Conclusion

The sophistication of recent attacks on UAVs [34], automobiles [20, 10], medical devices [12] as well as an industrial control systems [15], indicates that RTS are becoming more vulnerable. In this paper we are making steps towards the development of a comprehensive framework to integrate security mechanisms and provide a glimpse of *security design metrics* for RTS. Designers of RTS are now able to improve their security posture, which will also improve overall *safety* – and that is essentially the main goal of such systems.

References

1 Marshall Abrams and Joe Weiss. Malicious control system cyber security attack case study–Maroochy Water Services, Australia. *McLean, VA: The MITRE Corporation*, 2008.

2 Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *SE Journal*, 8(5):284–292, 1993.

3 BeagleBone Black. `https://beagleboard.org/black`.

4 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *RTS Journal*, 30(1-2):129–154, 2005.

5 Enrico Bini and Anton Cervin. Delay-aware period assignment in control systems. In *IEEE RTSS*, pages 291–300, 2008.

6 Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Opt. & Eng.*, 8(1):67–127, 2007.

7 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

8 The Bro Network Security Monitor. `https://www.bro.org`.

9 Alan Burns and Robert Davis. Mixed criticality systems – a review. Technical report, University of York, 2013. [Online]. URL: `https://www-users.cs.york.ac.uk/~burns/review.pdf`.

10 Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Sec. Symp.*, 2011.

11 Chien-Ying Chen, Rakesh B. Bobba, and Sibin Mohan. Schedule-based side-channel attack in fixed-priority real-time systems. Technical report, University of Illinois, 2015. [Online]. URL: `http://hdl.handle.net/2142/88344`.

12 Shane S. Clark and Kevin Fu. Recent results in computer security for medical devices. In *MobiHealth*, pages 111–118, 2011.

13 Rob Davis and Alan Burns. An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In *IEEE RTNS*, 2008.

14 Ethical hacking and countermeasures: Secure network operating systems and infrastructures, 2017.

15 Nicolas Falliere, Liam O. Murchu, and Eric Chien. W32. Stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5:6, 2011.

16 FreeRTOS. `http://www.freertos.org`.

17 FTP Brute-force attack trace. `https://github.com/bro/bro/blob/master/testing/btest/Traces/ftp/bruteforce.pcap`.

18 Monowar Hasan, Sibin Mohan, Rakesh B. Bobba, and Rodolfo Pellizzoni. Exploring opportunistic execution for integrating security into legacy hard real-time systems. In *IEEE RTSS*, pages 123–134, 2016.

19 Monowar Hasan, Sibin Mohan, Rakesh B. Bobba, and Rodolfo Pellizzoni. A server model to integrate security tasks into fixed-priority real-time systems. In *IEEE CERTS*, pages 61–68, 2016.

20 Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *IEEE S&P*, pages 447–462, 2010.

21 Man Lin, Li Xu, Laurence T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. Static security optimization for real-time systems. *IEEE Trans. on Indust. Info.*, 5(1):22–37, 2009.

22 Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *JACM*, 20(1):46–61, 1973.

23 Xue Liu, Hui Ding, Kihwal Lee, Qixin Wang, and Lui Sha. ORTEGA: An efficient and flexible software fault tolerance architecture for real-time control systems. In *IEEE ECRTS*, pages 125–134, 2008.

24 Daniel Lo, Mohamed Ismail, Tao Chen, and G. Edward Suh. Slack-aware opportunistic monitoring for real-time systems. In *IEEE RTAS*, pages 203–214, 2014.

25 Sibin Mohan. Worst-case execution time analysis of security policies for deeply embedded real-time systems. *ACM SIGBED Review*, 5(1):8, 2008.

26 Sibin Mohan, Man-Ki Yoon, Rodolfo Pellizzoni, and Rakesh B. Bobba. Real-time systems security through scheduler constraints. In *IEEE ECRTS*, pages 129–140, 2014.

27 Sibin Mohan, Man-Ki Yoon, Rodolfo Pellizzoni, and Rakesh B. Bobba. Integrating security constraints into fixed priority real-time schedulers. *RTS Journal*, 52(5):644–674, 2016. doi:10.1007/s11241-016-9252-5.

28 A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Massachusetts Institute of Technology, 1983.

29 Almir Mutapcic, Kwangmoo Koh, Seungjean Kim, Lieven Vandenberghe, and Stephen Boyd. GGPLAB: a simple Matlab toolbox for geometric programming, 2006. URL: https://stanford.edu/~boyd/ggplab/.

30 Rodolfo Pellizzoni, Neda Paryab, Man-Ki Yoon, Stanley Bak, Sibin Mohan, and Rakesh B. Bobba. A generalized model for preventing information leakage in hard real-time systems. In *IEEE RTAS*, pages 271–282, 2015.

31 Saowanee Saewong, Ragunathan (Raj) Rajkumar, John P. Lehoczky, and Mark H. Klein. Analysis of hierarchical fixed-priority scheduling. In *IEEE ECRTS*, pages 173–181, 2002.

32 Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. on Comp.*, 39(9):1175–1185, 1990.

33 Shellcode on ARM architecture. http://shell-storm.org/shellcode.

34 Daniel P. Shepard, Jahshan A. Bhatti, Todd E. Humphreys, and Aaron A. Fansler. Evaluation of smart grid and civilian UAV vulnerability to GPS spoofing attacks. In *Proc. of the ION GNSS Meeting*, volume 3, 2012.

35 Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *IEEE RTSS*, pages 2–13, 2003.

36 Open Source Tripwire. https://github.com/Tripwire/tripwire-open-source.

37 UAV Control Codes. https://github.com/Khan-drone/flight-control.

38 Xenomai – Real-time framework for Linux. https://xenomai.org.

39 Tao Xie and Xiao Qin. Improving security for periodic tasks in embedded systems through scheduling. *ACM TECS*, 6(3):20, 2007.

40 Man-Ki Yoon, Jung-Eun Kim, Richard Bradford, and Lui Sha. Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling. In *DATE*, pages 1313–1318, 2013.

41 Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *IEEE RTAS*, pages 1–12, 2016.

42 Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems. In *IEEE RTAS*, pages 21–32, 2013.