Exploring Opportunistic Execution for Integrating Security into Legacy Hard Real-Time Systems

Monowar Hasan[*], Sibin Mohan[*], Rakesh B. Bobba[†] and Rodolfo Pellizzoni[‡]
[*]*Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA*
[†]*School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA*
[‡]*Dept. of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada*
*Email: {*mhasan11, *sibin}@illinois.edu, †rakesh.bobba@oregonstate.edu, ‡rodolfo.pellizzoni@uwaterloo.ca*

***Abstract*—Due to physical isolation as well as use of proprietary hardware and protocols, traditional real-time systems (RTS) were considered to be invulnerable to security breaches and external attacks. This assumption is being challenged by recent attacks that highlight vulnerabilities in RTS. Besides, a straightforward integration of security mechanisms might compromise the safety and predictability guarantees of such systems. In this paper, we focus on integrating security mechanisms into RTS (especially *legacy* RTS) and define a *metric* to measure the effectiveness of such integration. We combine opportunistic execution with hierarchical scheduling to maintain compatibility with legacy systems while still providing flexibility. The proposed approach is shown to increase the security posture of RTS without impacting their temporal (and hence, safety) constraints.**

I. Introduction

Real-time Systems (RTS) are everywhere. Embedded RTS are used for monitoring and controlling physical systems and processes in varied domains, *e.g.,* aircraft including Unmanned Aerial Vehicles (UAVs), submarines, other vehicles (both autonomous as well as manual), critical infrastructures (*e.g.,* power grids and water systems), spacecraft and industrial plants to name but a few. These systems rely on a variety of inputs for their correct operation and have to meet stringent safety and timing requirements. However, until recently, cyber-security considerations were an afterthought in the design of such systems. While fault-tolerance has been a design consideration, traditional fault-tolerance techniques that were designed to counter and survive random or accidental faults are not sufficient to deal with cyber-attacks.

Given the increasing cyber-attack risks in RTS, it is essential to integrate resilience against such attacks into the design of RTS. There is also a need to retrofit existing critical RTS with detection, survival and recovery mechanisms. The focus of this work is on integrating or retrofitting security mechanisms into *legacy* RTS. A legacy RTS is one where modification or perturbation of existing real-time tasks' parameters (such as run-times, period, task execution order, *etc.*) is not always feasible. Therefore, any security mechanisms that are introduced not only have to co-exist with legacy real-time tasks without violating their real-time and safety constraints but also the *parameters of such legacy tasks cannot be adjusted to accommodate the security tasks.*

This creates an apparent tension, especially so in the case of *legacy* systems – between security requirements (*e.g.,* having enough cycles for effective detection) and the timing and safety requirements. Not only must the security mechanisms work effectively but they must also not interfere with the deadlines of real-time tasks. For instance, any monitoring and detection mechanism has to be designed so that an adversary cannot easily evade it. This may require that the monitoring and detection tasks be run frequently. However, the stringent timing constraints in hard RTS introduce additional complexities for the implementation of such cyber-security mechanisms – the strict deadlines for the completion of periodic hard RTS may not allow for frequent execution of security mechanisms. Further, unlike in conventional IT settings, it may not be possible to execute the security tasks for arbitrary lengths of time.

Our goal is to improve RTS security posture by integrating security mechanisms *without* violating real-time constraints. The security routines could be any intrusion detection and recovery mechanism depending on the system requirements. As an example, let us consider an open source intrusion detection mechanism, Tripwire[1], that detects integrity violations in the system. It stores clean system state during initialization[2] and uses it later to detect intrusions by comparing the current system state against the stored clean values. As shown in Table I, the default configuration of Tripwire contains several tasks, *viz.,* protecting its own binary files, protecting system binary and library files, ensuring kernel and process integrity, *etc.* In this paper we will use Tripwire as an example of a security program that needs to be integrated into legacy RTS – the ideas presented here though apply more broadly.

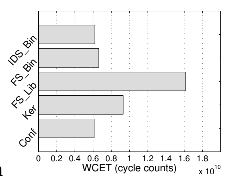
While integrating security mechanisms into a practical system, the following performance criteria need to be considered.

i) Monitoring Frequency: In order to provide the best protection, the security tasks need to be executed quite often. If the interval between consecutive monitoring events is too large, the adversary may harm the system

[1]http://www.tripwire.com/.

[2]This requirement of having an initial clean (trusted) system state is specific to the Tripwire example only. Our proposed framework, however, is independent of any such assumptions.

Table I
EXAMPLE OF SECURITY TASKS FOR INTRUSION DETECTION IN LINUX-BASED RTOS*

Task	Function	Criticality	Execution Time
Check IDS own binary (IDS_bin)	Scan (*e.g.,* compare hash value) files in the following locations: `/usr/sbin/siggen`, `/usr/sbin/tripwire`, `/usr/sbin/twadmin`, `/usr/sbin/twprint`	High	
Check critical executables (FS_bin)	Scan file-system binary (*e.g.,* `/bin`, `/sbin`)	High	
Check critical libraries (FS_lib)	Scan file-system library (*e.g.,* `/lib`)	High	
Check device and kernel (Ker)	Scan peripherals and kernel information in the `/dev` and `/proc` directory	High	
Check configuration files (Conf)	Scan changes in the configuration files (*e.g.,* `/etc`)	Medium	

*We use Linux kernel version 3.10.32 with real-time patch, *e.g.,* Real-Time Application Interface (RTAI) [1] version 4.1. The execution times are measured using read time-stamp counter (RDTSC) [2] instruction.

(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.

ii) *Responsiveness:* In some circumstances, a security task may need to execute with less interference from higher-priority tasks. For instance, let us consider the scenario where a security breach is suspected. In such an event the security task may be required to *perform more fine-grained checking instead of waiting for its next sporadic slot*. This may cause some low-priority non-critical real-time tasks to miss their deadlines. However, the scheduling policy needs to ensure that the system remains secure without violating real-time constraints for critical, high-priority, real-time tasks.

iii) *Atomicity:* Depending on the operation, 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.

In this paper we focus mainly on the *monitoring frequency* criterion[3]. However, we highlight how our proposed framework can be extended to incorporate *responsiveness* and *atomicity* in Section VIII. In particular, we consider incorporating security mechanisms by implementing them as separate *sporadic tasks*. This brings up the challenge of determining the 'right' periods (*e.g.,* minimum inter-monitoring time) 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. On the other hand, 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.

Therefore, we propose a framework that allows the execution of security tasks *opportunistically* with lower-priority than real-time tasks, while keeping the best possible periods for the security tasks that ensure all the tasks in the system remain schedulable. The main contribution of this paper can be summarized as follows:

- We introduce an extensible framework that allows the security tasks to execute without perturbing scheduling order and timing constraints of the real-time tasks (Section VI). In doing so, we formulate a constraint optimization problem and solve it using Geometric Programming (GP) approach in polynomial time (Sections III-V).
- We propose a *metric* to measure the security posture of the system in terms of frequency of periodic execution (Section III).
- We evaluate the proposed approach for schedulability and security (Section VII).

Key mathematical notations used in the paper are listed in Table II.

II. SYSTEM AND SECURITY MODEL

A. Real-Time Tasks and Scheduling Model

We consider a uni-processor system consisting of a set of m fixed-priority[4] sporadic real-time tasks $\Gamma_R = \{\tau_1, \tau_2, \cdots, \tau_m\}$. Each real-time task $\tau_i \in \Gamma_R$ is characterized by (C_i, T_i, D_i), where C_i worst-case execution time (WCET), T_i is the minimum inter-arrival time (or period) between successive releases and D_i is the relative deadline. We assume that priorities are distinct and the tasks have implicit deadline, *e.g.,* $D_i = T_i$ for $\forall i \in \Gamma_R$. For the simplicity of notation we use the same symbol τ_i to denote a task's jobs.

Let $hp(\tau_i)$ denote the set of tasks that have higher priority than τ_i. We assume that the real-time task-set Γ_R is

[3]As the other two imply changing the schedule of real-time tasks that, in most cases, is not feasible with legacy systems.

[4]We assume that task priority assignment follows Rate Monotonic (RM) [3] algorithm.

Table II
MATHEMATICAL NOTATIONS

Notation	Interpretation
Γ_R, Γ_S	Set of real-time and security tasks, respectively
m, n	Number of real-time and security tasks, respectively
C_i, D_i, T_i	Worst-case execution time, deadline, and period of task τ_i, respectively
$hp(\tau_i)$	Set of tasks that has higher priority than τ_i
T_i^{des}, T_i^{max}	Desired and maximum allowable period of the security task $\tau_i \in \Gamma_S$, respectively
ω_i	Weighting factor of the security task $\tau_i \in \Gamma_S$
$\mathcal{S}(Q, P)$	Security server with capacity Q and replenishment period P
$\tau_{\mathcal{S}}$	The task represents the security server $\mathcal{S}(Q, P)$ with capacity Q and replenishment period P
$UB_{\mathcal{S}(Q,P),\Gamma_S}$	Utilization bound for the given server parameters $\mathcal{S}(Q, P)$ and security task-set Γ_S
$\Delta_{\mathcal{S}}$	Worst case interference to the security server by the real-time tasks
I_i	Worst-case workload generated by the τ_i and $hp(\tau_i)$ from critical instance to deadline of τ_i
$\mathbf{lsbf}_{\mathcal{S}}(t)$	Linear lower-bound supply function of the security server during time interval t

schedulable by a fixed-priority preemptive scheduling. Since the task-set is schedulable, for each task $\tau_i \in \Gamma_R$, the worst-case response time w_i is less than or equal to the deadline D_i. Hence, the following inequity is satisfied for all tasks $\tau_i \in \Gamma_R$ [4], [5]:

$$w_i = C_i + \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{r_i^k}{T_h} \right\rceil \cdot C_h \leq D_i \tag{1}$$

where $r_i^0 = C_i$ and $w_i = r_i^{k+1} = r_i^k$ for some k.

B. Security Model

RTS face threats in various forms depending on the system and the goals of an adversary. For instance, adversaries may eavesdrop on, insert or modify messages exchanged by RTS; they may manipulate the sensor inputs being processed; the adversary could try to modify the control flow of the system; glean sensitive information through side channels; *etc.* The goals of adversaries may range from simply lodging themselves in the system and stealthily learning sensitive information to actively taking control and manipulating or crashing the system.

Threats to communications are usually dealt with by integrating cryptographic protection mechanisms. From an RTS perspective this increases the WCET of existing real-time tasks and has been studied in literature [6], [7]. In contrast, our focus is on threats that can be dealt with by integrating additional security tasks. For example, a sensor measurement correlation task may be added for detecting sensor manipulation or a change detection task may be added to detect changes or intrusions into the system. The addition of such tasks may necessitate changing the schedule of real-time tasks as was the case in [8], [9] where a state cleansing task was added to deal with stealthy adversaries gleaning sensitive information through side channels. In this work we focus on situations where added security tasks are not allowed to impact the schedule of existing real-time tasks as is often the case when integrating security into legacy systems. However, we do discuss the extensiblity of our approach to scenarios when some changes may be allowed.

C. Security Tasks

Our goal is to ensure security of the system *without* perturbing the scheduling order and timing constraint (*e.g.,* Eq. (1)) of the real-time tasks. As mentioned earlier, we ensure security of the system by integrating additional security tasks. Let us denote the security tasks by the set $\Gamma_S = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Each security task $\tau_i \in \Gamma_S$ is characterized by $(C_i, T_i^{des}, T_i^{max}, \omega_i)$, 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 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 given weighting factor, that may reflect the criticality of the security task τ_i. More critical security tasks would have larger w_i. For example, as illustrated in Table I, the default configuration of Tripwire has different criticality levels, *e.g., High* (critical files that are significant points of vulnerability), *Medium* (non-critical files that are of significant security impact), *etc.*

Any period T_i within the range $T_i^{des} \leq T_i \leq T_i^{max}$ would be acceptable. The actual period T_i, however, is not known a priori and our goal is to find the suitable period for the security tasks without violating the real-time constraints. We assume that the security tasks also have implicit deadline, *e.g.,* $D_i = T_i$ for $\forall \tau_i \in \Gamma_S$ that implies security tasks should complete before its next period.

III. PERIOD ADAPTATION

A simple approach to ensure security without perturbing real-time scheduling order is to execute security tasks as lower priority than real-time tasks. Hence, the security routines will be executing only during slack times when no other higher-priority real time tasks are running. However, one fundamental problem is to determine *which* security tasks will be running *when* to provide better defense against certain vulnerabilities. As mentioned earlier, actual period of the security tasks is unknown and we need to *adapt* the periods within acceptable ranges to achieve better trade-off between schedulabily and defense against security breaches. In what follows, we formulate the period adaptation problem to ensure security tasks can execute close to their desired frequency.

A. Problem Description

We measure the security of the system by means of the *achievable periodic monitoring*. Let T_i be the period of the security task $\tau_i \in \Gamma_S$ that needs to be determined. Our goal is to minimize the perturbation between the achievable period T_i and the desired period T_i^{des}. Hence we define the 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 and bounded by $\frac{T_i^{des}}{T_i^{max}} \leq \eta_i \leq 1$. The period adaptation problem is therefore to maximize the tightness of the achievable periods of all the security tasks without violating the schedulability as well as period bound constraints of the security tasks. The formulation of the period adaptation problem is explained in the following.

B. Formulation as an Optimization Problem

1) Objective Function: As mentioned earlier, the objective of the period adaptation is to minimize the perturbation (*e.g.,* maximize the tightness η_i) for all the security tasks. Mathematically the objective function can be defined as follows:

$$\max_{\mathbf{T}} \eta \tag{3}$$

where $\eta = \sum\limits_{\tau_i \in \Gamma_S} \omega_i \eta_i = \sum\limits_{\tau_i \in \Gamma_S} \omega_i \frac{T_i^{des}}{T_i}$ is the cumulative weighted tightness and the vector $\mathbf{T} = [T_1, T_2, \cdots, T_n]^\intercal$ represents the period of the tasks that need to be determined.

2) Utilization Bound Constraint: Utilization of each security task is an important measure since it reveals the amount of processor dedicated to the security task and thus impacts schedulability. Since we are executing security tasks in the idle time, the following necessary condition ensures that utilization of the security tasks are within the remaining utilization [3]:

$$\sum_{\tau_i \in \Gamma_S} \frac{C_i}{T_i} \leq (m+n)(2^{\frac{1}{m+n}} - 1) - \sum_{\tau_j \in \Gamma_R} \frac{C_j}{T_j}. \tag{4}$$

3) Period Bound Constraints: Since we consider that the integrated system consisting of real-time and security tasks will follow the RM priority order, the following constraint needs to be satisfied

$$T_i \geq \bar{T} \quad \forall \tau_i \in \Gamma_S \tag{5}$$

where $\bar{T} = \max\limits_{\tau_j \in \Gamma_R} T_j$. Besides, in order to fulfill the restrictions on periodic monitoring, the following inequality needs to be satisfied for all the security tasks

$$T_i^{des} \leq T_i \leq T_i^{max} \quad \forall \tau_i \in \Gamma_S. \tag{6}$$

Using the objective function in Eq. (3), and the set of constraints in Eqs. (4), (5), and (6), we can formulate the period adaptation problem as the following non-linear constraint optimization problem.

$$(\mathbf{P1}) \ \max_{\mathbf{T}} \sum_{\tau_i \in \Gamma_S} \omega_i \frac{T_i^{des}}{T_i}, \quad \text{Subject to: } (4), (5), (6).$$

Although it is non-trivial to solve the above non-linear non-convex optimization problem in its current form, it is possible to transform $\mathbf{P1}$ into a convex optimization problem using an approach similar to that presented in this paper. However, the analysis using the above formulation is limited by RM bound and also the security task's periods need to satisfy the constraint in Eq. (5) to follow RM priority order. In addition, rather than only focusing on optimizing the periods of the security tasks, we aim to design a *unified* framework that can achieve other security aspects (*e.g.,* responsiveness and atomicity). Hence, instead of simply running security tasks by themselves in idle-time, we propose using a *server* to execute the security tasks. With this approach, for instance, if better responsiveness is desired from security mechanisms, we could increase the priority of the server and allow the server to execute until the security task finishes its desired checking[5]. Not only will the server abstraction allow us to provide better isolation between real-time and security tasks; but it also enables us to integrate additional security properties and provide better execution frequency for certain conditions as we discuss in Sections VII and VIII.

IV. The Security Server

The security server is an abstraction that provides execution time to the security tasks, according to a preemptive fixed-priority scheduling algorithm (*e.g.,* RM). Instead of focusing on any particular server algorithm, we consider a generic server abstraction model. The security server $\mathcal{S}(Q, P)$ is characterized by the *capacity* Q and *replenishment period* P and works as follows.

The server may be in two states, *e.g., active* and *inactive* and executed with *lowest-priority*. If any security task is activated at time t and if the server is inactive, then the server will become active with capacity Q and relative deadline (*e.g.,* next replenishment time) is set as $t + P$. If the server is already active, then the current capacity and relative deadline remain unchanged. When the server is being scheduled, it executes the security tasks according to its own scheduling policy; which we consider fixed-priority RM scheduling in this work. While a security task is executing, the current available capacity is decremented accordingly. The server can be preempted by the scheduler in order to serve the real-time tasks. When the server is preempted, the currently available capacity is no longer decremented. If the available capacity becomes zero and some security task has not yet finished, then the server is suspended until its next replenishment time. Let t' be that replenishment time. At time t', the server is recharged to its full capacity Q, the next replenishment time is set as $t' + P$ and the server can execute again. When the last security task has finished executing and there is no other pending task in the server, the server will be suspended. Also, the server will become inactive if there are no security tasks ready to execute.

There is no strict assumption on the smallest time unit of server parameters, *e.g.,* $Q, P \in \mathbb{R}^+$ and the security task releases are not bound to the server [10]. Besides, we assume that there is no task or server release jitters.

A. Reformulation of Period Adaptation Problem using Server

Since we execute the security tasks within the server, the constraint in Eq. (4) needs to be revised accordingly to

[5]This issue is discussed further in Section VIII.

consider the server's available capacity and replenishment period. Let us denote $UB_{\mathcal{S}(Q,P),\Gamma_S}$ as the utilization bound for the set of security task Γ_S executing within the server with capacity Q and replenishment period[6] P. Since the server schedules the security tasks on a fixed-priority order, we can determine the utilization bound $UB_{\mathcal{S}(Q,P),\Gamma_S}$ using the concept similar to that discussed in literature [11]. The authors in work [11] represented the utilization bound as a function of number of tasks running under the server; which is essentially derived from the Liu and Layland's utilization bound [3]. In particular, when the smallest period of the security task is greater than or equal to $3P-2Q$, the upper bound of the utilization factor for the security tasks is given by $UB_{\mathcal{S}(Q,P),\Gamma_S} = n\left[\left(\frac{3-\frac{Q}{P}}{3-2\frac{Q}{P}}\right)^{\frac{1}{n}} - 1\right]$ where n is number of security tasks in the set Γ_S. Hence we can define the following constraints on utilization bound

$$\sum_{\tau_i \in \Gamma_S} \frac{C_i}{T_i} \leq UB_{\mathcal{S}(Q,P),\Gamma_S} = n\left[\left(\frac{3-\frac{Q}{P}}{3-2\frac{Q}{P}}\right)^{\frac{1}{n}} - 1\right] \quad (8)$$

$$T_i \geq 3P - 2Q \quad \forall \tau_i \in \Gamma_S. \quad (9)$$

Therefore, the period optimization problem with presence of server can be presented similar to that of **P1** except the constraints in Eqs. (4) and (5) will be replaced by the constraints in Eqs. (8) and (9).

B. Geometric Programming (GP) Formulation

Solving the above constraint non-linear problem is not straightforward and it always involves compromise of accepting a local instead of global solution. Therefore we reformulate the optimization problem as a GP. The GP reformulation allows us to solve the problem in an efficient way. When the constraints are mutually consistent, the GP solution approach can always finds a globally optimal solution [12]. In what follows, we first introduce the GP framework and then show how the period adaptation problem can be cast as a GP.

1) Geometric Programming: A non-linear optimization problem can be solved by GP if the problem is formulated as follows [12]

$$\min_{\mathbf{x}} \; f_0(\mathbf{x}) \quad (10a)$$

$$\text{Subject to: } f_i(\mathbf{x}) \leq 1 \quad i = 1, \cdots, z_p \quad (10b)$$

$$g_i(\mathbf{x}) = 1 \quad i = 1, \cdots, z_m \quad (10c)$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_z]^\mathsf{T}$ denotes the vector of z optimization variables. The functions $f_0(\mathbf{x}), f_1(\mathbf{x}), \cdots, f_{z_p}(\mathbf{x})$ are *posynomial* and $g_1(\mathbf{x}), \cdots, g_{z_m}(\mathbf{x})$ are *monomial* functions, respectively. A function $g_i(\mathbf{x})$ is monomial if it can be expressed as $g_i(\mathbf{x}) = c_i \prod_{l=1}^{L_i} x_l^{a_l}$ where $c_i \in \mathbb{R}^+$ and $a_l \in \mathbb{R}$. 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 is the sum of the monomials, and thus can be represented as $f_i(\mathbf{x}) = \sum_{l=1}^{L_i} c_l x_1^{a_{1l}} x_2^{a_{2l}} \cdots x_z^{a_{1l}}$ where $c_l \in \mathbb{R}^+$ and $a_{jl} \in \mathbb{R}$. The posynomials are closed under addition, multiplication and non-negative scaling where the monomials are closed under multiplication and division.

[6]The calculation of capacity and replenishment period is discussed in Section V.

2) Period Adaptation as a GP: In the following we reformulate the period adaptation problem as a GP.

Observation 1. The fundamental measures, *e.g.,* $\sum_{\tau_i \in \Gamma_S} \omega_i \frac{T_i^{des}}{T_i} = \sum_{\tau_i \in \Gamma_S} \omega_i T_i^{des} T_i^{-1}$ and $\sum_{\tau_i \in \Gamma_S} \frac{C_i}{T_i} = \sum_{\tau_i \in \Gamma_S} C_i T_i^{-1}$ in the period adaptation problem are posynomials.

This is directly follows from the observation that all the coefficients are non-negative and the variables (*e.g.,* periods) are always positive. Besides, we are only summing up positive terms and therefore the terms are closed under addition. Since the requirement for posynomials is that it need to be closed under addition, the above terms are posynomials.

An interesting property of posynomials and monomials is that, if $f(\cdot)$ is a posynomial and $g(\cdot)$ is a monomial, the ratio $\frac{f(\cdot)}{g(\cdot)}$ will become a posynomial. Since $\frac{f(\cdot)}{g(\cdot)}$ is a posynomial, this allows us to express the constraint $f(\cdot) < g(\cdot)$ as $\frac{f(\cdot)}{g(\cdot)} \leq 1$. For example, we can easily handle the constraint of the form $f(\cdot) \leq \alpha$ where $f(\cdot)$ is a posynomial and $\alpha > 0$. We can refer $\hat{f}(\cdot)$ is an *inverse posynomial* if $\frac{1}{\hat{f}(\cdot)}$ is a posynomial. Besides, we can maximize a non-zero posynomial objective function by minimizing its inverse. Based on the above description, we can reformulate the maximization problem as a standard GP minimization problem as follows

$$(\mathbf{P2}) \qquad \min_{\mathbf{T}} \sum_{\tau_i \in \Gamma_S} {\omega_i}^{-1} (T_i^{des})^{-1} T_i \quad (11a)$$

Subject to:

$$\Big(\sum_{\tau_i \in \Gamma_S} C_i {T_i}^{-1}\Big) \cdot \big(UB_{\mathcal{S}(Q,P),\Gamma_S}\big)^{-1} \leq 1 \quad (11b)$$

$$(3P-2Q){T_i}^{-1} \leq 1 \; \forall \tau_i \in \Gamma_S \quad (11c)$$

$$T_i^{des} {T_i}^{-1} \leq 1 \; \forall \tau_i \in \Gamma_S \quad (11d)$$

$$(T_i^{max})^{-1} T_i \leq 1 \; \forall \tau_i \in \Gamma_S \quad (11e)$$

where $\big(UB_{\mathcal{S}(Q,P),\Gamma_S}\big)^{-1} = \frac{1}{n\left[\left(\frac{3-\frac{Q}{P}}{3-2\frac{Q}{P}}\right)^{\frac{1}{n}} - 1\right]}$.

The GP formulation **P2** is not a convex optimization problem since the posynomials are not convex functions [12]. However, **P2** can be converted into a convex optimization problem using logarithmic transformations. The conversion **P2** into a convex optimization problem is based on a logarithmic change of variables, as well as a logarithmic transformation of objective and constraint functions. Instead of using optimization variable T_i let us use the logarithms, *e.g.,* $\tilde{T}_i = \log T_i$ and hence $T_i = e^{\tilde{T}_i}$. Besides, let us replace inequality constraints of the form $f_i(\cdot) \leq 1$ with $\log f_i(\cdot) \leq 0$. Using this transformation we can express $\mathbf{P2}$ as follows

$$(\mathbf{P3}) \qquad \min_{\tilde{\mathbf{T}}} \ \log \sum_{\tau_i \in \Gamma_S} e^{{\omega_i}^{-1}(T_i^{des})^{-1}\tilde{T}_i} \tag{12a}$$

Subject to:

$$\log e^{\left(\sum\limits_{\tau_i \in \Gamma_S} C_i \tilde{T}_i^{-1}\right)\cdot\left(UB_{\mathcal{S}(Q,P),\Gamma_S}\right)^{-1}} \leq 0 \tag{12b}$$

$$\log e^{(3P-2Q)\tilde{T}_i^{-1}} \leq 0 \ \forall \tau_i \in \Gamma_S \tag{12c}$$

$$\log e^{T_i^{des}\tilde{T}_i^{-1}} \leq 0 \ \forall \tau_i \in \Gamma_S \tag{12d}$$

$$\log e^{(T_i^{max})^{-1}\tilde{T}_i} \leq 0 \ \forall \tau_i \in \Gamma_S \tag{12e}$$

where $\tilde{T}_i = \log T_i$ and $\tilde{\mathbf{T}} = [\tilde{T}_1, \tilde{T}_2, \cdots \tilde{T}_n]^\mathsf{T}$. This logarithmic transformation leads $\mathbf{P3}$ to a convex optimization problem with respect to new variable $\tilde{\mathbf{T}}$ [12] [13, Secs. 4.5 and 3.1.5]. Since $\mathbf{P3}$ is a convex optimization problem, it can be solved using standard algorithms such as *interior-point* method that is known to be solvable in polynomial time [13, Ch. 11].

V. Selection of the Server Parameters

The period adaptation problem in the preceding section is derived based on a given server parameter $\mathcal{S}(Q,P)$ *e.g.,* the utilization bound $UB_{\mathcal{S}(Q,P),\Gamma_S}$ in constraint (12c). However, one fundamental problem is to find a suitable pair of server capacity Q and replenishment period P that respects the real-time constraints of the tasks in the system. In this section, for a given period of the security tasks, we formulate a GP problem to determine the server parameters.

A. Linear Lower-Bound Supply Function

For the security server with unknown capacity Q and replenishment period P, we can derive the lower (upper) bound of Q (P) that makes security tasks $\tau_i \in \Gamma_S$ running under server schedulable by using periodic server model introduced in literature [14] [15] [16]. The key idea from previous work is that a task τ_i can be schedulable if minimum supply for the server can match the maximum workload generated by τ_i and $hp(\tau_i)$ during a time interval t. If the server task $\tau_{\mathcal{S}}$ is scheduled by a fixed-priority scheme, the minimum supply of the server is delivered to the security tasks when its $(k-1)$-th execution has just finished with minimum interference from the high-priority real-time tasks $\tau_j \in \Gamma_R$. Then, the subsequent executions of k-th release are maximally delayed by the higher-priority real-time tasks. For this minimum supply, we can parameterize the *linear lower-bound supply function* $\mathbf{lsbf}_{\mathcal{S}}(t)$ with the period and WCET of higher-priority real-time tasks.

The worst-case response time of the server is the longest time from the server being replenished to its capacity being exhausted with the maximum interference from the high-priority real-time tasks, given that there are security tasks ready to use all of the server's available capacity. In order to calculate exact response time of the server, we can use the formula introduced in the work [4]. Using this exact method, we can calculate the maximum possible preemption on the server from the higher-priority real-time tasks for a certain length of window and add up the server's capacity. The calculation is repeated iteratively by increasing the window size until the window size exceeds the server's relative replenishment period (in this case the system is determined to be unschedulable) or until the window size is stable. Then, the window size is the *busy period* and let us denote it as $w_{\mathcal{S}}$.

The worst-case release pattern of server occurs when $\tau_{\mathcal{S}}$ and $hp(\tau_{\mathcal{S}})$ is released simultaneously. The worst-case busy period $w_{\mathcal{S}}$ is the maximum time duration that the server can take to execute full capacity Q when it is released simultaneously with the higher-priority real-time tasks, $hp(\tau_{\mathcal{S}})$ at the k-th release. Therefore, by using the traditional exact analysis [4] the worst-case busy period can be obtained as

$$w_{\mathcal{S}}^{k+1} = Q + \sum_{\tau_h \in hp(\tau_{\mathcal{S}})} \left\lceil \frac{w_{\mathcal{S}}^k}{T_h} \right\rceil \cdot C_h \tag{13}$$

where $w_{\mathcal{S}}^0 = Q$ and $w_{\mathcal{S}} = w_{\mathcal{S}}^{k+1} = w_{\mathcal{S}}^k$ when it converges for some k. Therefore the worst-case delay at the k-th release and thereafter can be represented as

$$\Delta_{\mathcal{S}} = \sum_{\tau_h \in hp(\tau_{\mathcal{S}})} \left\lceil \frac{w_{\mathcal{S}}}{T_h} \right\rceil \cdot C_h. \tag{14}$$

However, such iterative methods are only amenable to brute-force approach. This is because, the ceiling function with unknown value (*e.g.,* the busy period) can not be in our formulation. Thus we take a different approach by approximating $\Delta_{\mathcal{S}}$. During a time interval of P, the maximum workload generated by the server and higher-priority real-time tasks can be represented by

$$w_{\mathcal{S}} = Q + \sum_{\tau_h \in hp(\tau_{\mathcal{S}})} \left\lceil \frac{P}{T_h} \right\rceil \cdot C_h. \tag{15}$$

Thus using Eq. (15), we can avoid the iterative calculation by assuming the number of invocation of higher-priority real-time tasks during P, not during the exact busy period of the server. Since $\lceil y \rceil \leq y + 1$, we linearize $w_{\mathcal{S}}$ by removing the ceiling function and represent Eq. (15) as

$$w_{\mathcal{S}} = Q + \sum_{\tau_h \in hp(\tau_{\mathcal{S}})} \left(\frac{P}{T_h} + 1\right) \cdot C_h. \tag{16}$$

Therefore, the worst-case linear lower-bound supply function of the security server during a time interval t [14] is given by

$$\mathbf{lsbf}_{\mathcal{S}}(t) = \frac{Q}{P}\left[t - (P - Q) - \Delta_{\mathcal{S}}\right] \tag{17}$$

where $\Delta_{\mathcal{S}} = \sum\limits_{\tau_h \in hp(\tau_{\mathcal{S}})} \left(\frac{P}{T_h} + 1\right) \cdot C_h$.

B. Sufficient Bound for Schedulability of the Security Tasks

In order to derive the minimum capacity that guarantees to schedule $\tau_i \in \Gamma_S$, let us consider the situation when

τ_i barely meets it deadline at $t = D_i$ with the worst-case interference from high-priority security tasks, $hp(\tau_i) \in \Gamma_S$. Let us now define the *critical instant* of the security tasks, *e.g.,* the worst-case response time of τ_i when τ_i and $hp(\tau_i)$ are released simultaneously at the end of server's $(k-1)$-th execution and suffer worst-case preemptions from k-th release and thereafter [16]. Let us denote I_i as the worst-case workload generated by the τ_i and $hp(\tau_i)$ from critical instant to deadline of τ_i given by

$$I_i = C_i + \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{D_i}{T_h} \right\rceil \cdot C_h. \quad (18)$$

In order to ensure the schedulability of the security task τ_i, the minimum supply delivered by the server has to be greater than or equal to the worst-case workload during the time interval D_i, *e.g.,*

$$\mathbf{lsbf}_{\mathcal{S}}(D_i) \geq I_i \quad \forall \tau_i \in \Gamma_S \quad (19)$$

where $\mathbf{lsbf}_{\mathcal{S}}(\cdot)$ is given by Eq. (17).

It is worth noting that Eq. (19) is only a sufficient and not necessary condition. The security task τ_i can be schedulable if and only if there exists a time instance $t \leq D_i$ such that the inequality in Eq. (19) holds. However, we use the sufficient condition in Eq. (19) because the presence of time in the necessary condition makes the proposed optimization framework inapplicable to the problem under consideration. Despite the fact that this bound may not be exact and may incur approximation error in the supply function, as we show in Sections VII and VIII, it enables us to ensure opportunistic execution of the security tasks without violating real-time constraints.

C. Formulation as an Optimization Problem

In the following we present the optimization problem formulation to determine the server parameters Q and P.

1) Objective Function: The objective of periodic monitoring in the system is to ensure maximal processor utilization for the security tasks, without violating the real-time constraints of the system. Therefore, we define the objective function as follows

$$\max_{Q,P} \frac{Q}{P} \quad (20)$$

where the server parameters, Q and P are the optimization variables.

2) Server Schedulability Constraint: A server is schedulable if worst-case response time of the server does not exceed its replenishment period [10]. Hence we represent the server schdulability constraint as follows

$$Q + \sum_{\tau_h \in hp(\tau_S)} \left(\frac{P}{T_h} + 1 \right) \cdot C_h \leq P. \quad (21)$$

3) Server Bound Constraints: As we have discussed in Section V-B, in order to guarantee schedulability of each of the security tasks $\tau_i \in \Gamma_S$, the linear lower-bound supply function must be greater than or equal to worst-case workload during the time interval D_i. Therefore, from Eq. (19) the constraints on the server supply bound to ensure schedulability of the security tasks can be expressed as

$$\frac{Q}{P}\left[D_i - (P - Q) - \Delta_{\mathcal{S}}\right] \geq I_i \quad \forall \tau_i \in \Gamma_S \quad (22)$$

where $T_i^{des} \leq D_i = T_i^* \leq T_i^{max}$ and T_i^* is obtained by solving the period adaptation problem $\mathbf{P3}$. Notice that, in Eq. (22) $I_i = C_i + \sum\limits_{\tau_h \in hp(\tau_i)} \left\lceil \frac{T_i^*}{T_h} \right\rceil \cdot C_h$ is the worst-case workload generated by τ_i and $hp(\tau_i)$ during the time interval of T_i^*. This is a constant for a given input.

D. Geometric Programming Formulation

Proposition 1. The objective function in Eq. (20) and the constraint in Eq. (21) can be expressed in posynomial form.

Proof: The proof follows by rearranging the terms in posynomial form and transform the objective function in Eq. (20) into minimization expression. Let us rearrange Eq. (20) as QP^{-1} which is clearly a posynomial. In order to reform the objective function in Eq. (20) as a standard GP minimization problem, we can rewrite Eq. (20) as

$$\min_{Q,P} Q^{-1}P \quad (23)$$

which is also in posynomial form. Let us now rearrange Eq. (21) as follows

$$(Q + \Delta_S)P^{-1} \leq 1 \quad (24)$$

where $\Delta_S = \sum\limits_{\tau_h \in hp(\tau_S)} (P + T_h) \cdot T_h^{-1} \cdot C_h$. Since the optimization variables (*e.g.,* capacity and replenishment period) are always positive, using the similar argument presented in Observation 1, we can assert that Eq. (24) is a posynomial constraint. ■

With a view to expressing the server bound constraint as posynomial form, we can rearrange Eq. (22) as follows

$$\frac{P(Q + I_i) + \Delta_S Q}{Q(Q + T_i^*)} \leq 1 \quad \forall \tau_i \in \Gamma_S. \quad (25)$$

Recall that, in order to represent the constraint of the form $\frac{f(\cdot)}{g(\cdot)} \leq 1$ the denominator must be monomial. However, the inequality in (25) does not conform to a posynomial form due to the posymolial term in the denominator, *e.g.,* $Q(Q + T_i^*) = Q^2 + QT_i^*$. The following theorem illustrates how the constraints on server bound can be represented in posymonial form.

Theorem 1. The server bound constraints can be formulated as the following posynomial form

$$[P(Q + I_i) + \Delta_S Q] \cdot [Q \cdot \hat{g}(Q, T_i^*)]^{-1} \leq 1 \, \forall \tau_i \in \Gamma_S. \quad (26)$$

Proof: The theorem is proved by using the geometric mean approximation [17, Ch. 2] of posynomials. Since the denominator in Eq. (25) is a posynomial, let us approximate $Q + T_i^*$ with a monomial by the following geometric mean approximation.

Let us denote $Q + T_i^*$ as $g(Q, T_i^*) = u_1(Q) + u_2(T_i^*)$ where $u_1(Q) = Q$ and $u_2(T_i^*) = T_i^*$. We can approximate $g(Q, T_i^*)$ with

$$\hat{g}(Q, T_i^*) = \left[\frac{u_1(Q)}{a}\right]^a \cdot \left[\frac{u_2(T_i^*)}{b}\right]^b \quad (27)$$

where $a = \frac{u_1(y_0)}{g(y_0, T_i^*)}$, $b = \frac{u_2(T_i^*)}{g(y_0, T_i^*)}$ and $y_0 \in \mathbb{R}^+$ is a constant that satisfies $\hat{g}(y_0, T_i^*) = g(y_0, T_i^*)$. The approximated monomial $\hat{g}(Q, T_i^*)$ can be rewritten as

$$\hat{g}(Q, T_i^*) = \left(\frac{Q}{a}\right)^a \cdot \left(\frac{T_i^*}{b}\right)^b \tag{28}$$

where $a = \frac{y_0}{y_0 + T_i^*}$, $b = \frac{T_i^*}{y_0 + T_i^*}$. Using this monomial approximation, we can represent Eq. (25) as

$$\frac{P(Q + I_i) + \Delta_S Q}{Q \cdot \hat{g}(Q, T_i^*)} \le 1 \quad \forall \tau_i \in \Gamma_S \tag{29}$$

and the proof follows.

■

Using the logarithmic transformation presented in Section IV-B2, we can formulate server parameter selection problem as a GP in convex form as follows

$$(\mathbf{P4}) \qquad \min_{\tilde{Q}, \tilde{P}} \ \log e^{\tilde{Q}^{-1}\tilde{P}} \tag{30a}$$

Subject to:

$$\log e^{\left[\tilde{Q} + \sum\limits_{\tau_h \in hp(\tau_S)} (\tilde{P} + T_h) \cdot T_h^{-1} \cdot C_h\right] \cdot \tilde{P}^{-1}} \le 0 \tag{30b}$$

$$\log e^{\left[\tilde{P}(\tilde{Q} + I_i) + \Delta_S \tilde{Q}\right] \cdot \left[\tilde{Q} \cdot \hat{g}(\tilde{Q}, T_i^*)\right]^{-1}} \le 0 \ \forall \tau_i \in \Gamma_S \tag{30c}$$

where $\tilde{Q} = \log Q$ and $\tilde{P} = \log P$. Since $\mathbf{P4}$ is a convex optimization problem, it is solvable using standard algorithm such as interior-point method.

VI. Algorithm Development

We develop an iterative scheme to obtain the period of the security tasks and the server parameters jointly. The overall procedure, as summarized in **Algorithm 1** works as follows. The algorithm starts with the maximum allowable periods (*e.g.,* T_i^{max}) and try to find the Q and P by solving the server parameter selection problem $\mathbf{P4}$. If there is no solution, which implies that the constraints in $\mathbf{P4}$ are mutually inconsistent, and it is not possible to provide a server for the security tasks without violating the real-time constraints of the system. Since it is then not possible to integrate security tasks in the given system, the algorithm reports the set of real-time and security tasks as unschedulable.

If there is a solution, we iteratively estimate the best period for the security tasks (Lines 8-26). In particular, for the given server parameter $Q(j)$ and $P(j)$ for any iteration j, the algorithm estimates the periods for next iteration by solving $\mathbf{P3}$. Likewise, for a given period vector, we calculate the best server parameters that make the constraints on $\mathbf{P4}$ mutually consistent. Let $\eta(j)$ be the objective value by solving $\mathbf{P3}$ at iteration j. The iteration is repeated as long as the difference of objective value in successive iterations is greater than some predefined tolerance for convergence ϵ, *e.g.,* $|\eta(j) - \eta(j-1)| > \epsilon$ or the maximum allowable iteration counter J_{max} is not exceeded. From our experiments we find that, the algorithm generally converged within 3-5 iterations for most of the schedulable task-sets with tolerance $\epsilon = 10^{-16}$.

Recall that, the approximation quality of $\hat{g}(\tilde{Q}, T_i^*)$ (*e.g.,* constraint (30c) in $\mathbf{P4}$) depends on the choice of y_0.

Algorithm 1 Security Period and Server Parameter Selection

Input: Set of real-time and security tasks, Γ_R and Γ_S, respectively
Output: The tuple $\{\mathbf{T}^*, Q^*, P^*\}$, *e.g.,* periods of the security tasks and the server parameters if the task-set is schedulable; Unscheduable otherwise

```
1: Initialize $j := 1$
2: Initialize security task's period vector $\mathbf{T}(j) := [T_i^{max}]^{\mathsf{T}}_{\forall \tau_i \in \Gamma_S}$
3: For the given $\mathbf{T}(j)$, Solve $\mathbf{P4}$ to obtain server parameters
4: if no solution found then
5:     /* not possible to integrate security tasks in the system */
6:     return Unscheduable
7: else
8:     Set $Q(j) = Q_*$, $P(j) = P_*$ where $Q_*, P_*$ be the solution from $\mathbf{P4}$
9:     while period perturbations are not minimized and $j < J_{max}$ do
10:        Update $j := j + 1$
11:        Solve the period adaptation problem $\mathbf{P3}$ using server parameters $Q(j-1), P(j-1)$
12:        if no solution found then
13:            /* return current best solution */
14:            Set $\mathbf{T}(j) := \mathbf{T}(j-1), Q(j) := Q(j-1), P(j) := P(j-1)$
15:            break
16:        end if
17:        Update the period vector $\mathbf{T}(j) := \mathbf{T}_*$ where $\mathbf{T}_*$ is the solution obtained from $\mathbf{P3}$
18:        For the given $\mathbf{T}(j)$, Solve $\mathbf{P4}$ to obtain server parameters
19:        if no solution found then
20:            /* return current best solution */
21:            Set $Q(j) := Q(j-1), P(j) := P(j-1)$
22:            break
23:        end if
24:        Update the server parameters $Q(j) := Q_*, P(j) := P_*$
25:    end while
26:    return $\{\mathbf{T}^* := \mathbf{T}(j), Q^* := Q(j), P^* := P(j)\}$
27: end if
```

Thus, in the optimization procedure (*e.g.,* Line 3 and 18), we iteratively approximate $\hat{g}(\tilde{Q}, T_i^*)$ by updating a and b according to the intermediate solution of $\tilde{Q}$. That is, until the objective value converges, we use $\tilde{Q}$ at k-th step as y_0 at $(k+1)$-th step. In our experiments, we choose the initial value of y_0 as 1, and the objective value of $\mathbf{P4}$ converged within 5 iterations.

VII. Evaluation

To evaluate the performance of our proposed opportunistic monitoring model, we performed experiments using randomly generated workloads as well as a prototype implementation on real-time Linux. The parameters used in our evaluation are summarized in Table III.

A. Evaluation Setup

The base-utilization of a task-set is defined as the total sum of the task utilizations. The real-time and security task-sets are grouped by base-utilization from $[0.01 + 0.1 \cdot i, 0.1 + 0.1 \cdot i]$ where $i \in \mathbb{Z}$ and the number of base-utilization groups are specified in the corresponding experiments. This allows us to generate task-sets with an even distribution of tasks. Each task-set instance contains $[3, 10]$ real-time and $[2, 5]$ security tasks. We assume that $\omega_i = 1$, $\forall \tau_i \in \Gamma_S$. Each real-time task $\tau_i \in \Gamma_R$ has a period $T_i \in [10ms, 100ms]$. The desired and maximum allowable periods for the security tasks

Table III
EXPERIMENTAL PARAMETERS

Parameter	**Values**
Number of real-time tasks, m	$[3, 10]$
Number of security tasks, n	$[2, 5]$
Real-time task period, T_i	$[10ms, 100ms]$
Desired period for security tasks, T_i^{des}	$[250ms, 500ms]$
Maximum allowable period, T_i^{max}	$[5000ms, 5050ms]$
Tolerance for convergence, ϵ	10^{-16}
Maximum replenishment period (for exhaustive search), P_{max}	2500
Exhaustive search granularity, δ	0.5

are selected from $[250ms, 500ms]$ and $[5000ms, 5050ms]$, respectively.

The utilization of the real-time and security tasks are generated by the UUniFast [18] algorithm. For a given utilization U_i, the execution time of τ_i is generated by $C_i = U_i T_i$. We use GGPLAB [19] to solve the GPs. All the experiments are performed on Intel Pentium N3530 2.16 GHz processor with 4 GB RAM.

B. Results

1) Comparison with Exact Method: We compare our GP-based approach with an exhaustive search method[7] based on exact analysis. In this exhaustive search, we assign server replenishment period from 1 to P_{max} with a granularity of δ. For each period, we determine the minimum capacity requirements that makes the tasks schedulable. From the set of feasible period and capacity pair, we take the pair that maximizes the server utilization. Notice that, the minimum server capacity $Q_{min}(\tau_i, P)$ for $\tau_i \in \Gamma_S$ with a given P can be obtained by solving the quadratic inequality in Eq. (22), which is given by

$$Q_{min}(\tau_i, P) = \frac{-(D_i - P - \Delta_{\mathcal{S}}) + \sqrt{(D_i - P - \Delta_{\mathcal{S}})^2 + 4 I_i P}}{2}. \quad (31)$$

In the above equation $\Delta_{\mathcal{S}}$ is calculated by exact method, *e.g.,* $\Delta_{\mathcal{S}} = \sum\limits_{\tau_h \in hp(\tau_{\mathcal{S}})} \left\lceil \frac{w_{\mathcal{S}}}{T_h} \right\rceil \cdot C_h$ where $w_{\mathcal{S}}$ is obtained from Eq. (13). In order to find the minimum required capacity of the server for a given replenishment period P, we take the maximum of the capacity $Q_{min}(\tau_i, P)$ over all the security tasks $\tau_i \in \Gamma_S$ which is defined as

$$Q_{min}(P) = \max_{\tau_i \in \Gamma_S} \left\{ Q_{min}(\tau_i, P) \right\}. \quad (32)$$

Hence any Q from $[Q_{min}(P), P]$ such that $Q + \Delta_{\mathcal{S}} \leq P$ will be the feasible capacity that makes the task-set schedulable.

In Fig. 1 we compare the number of schedulable task-sets found in the proposed method and exhaustive search. For exhaustive search, we set $P_{max} = 2500$ with granularity $\delta = 0.5$. As we can see from figure, the difference in terms of schchdulable task-sets found by exhaustive search compared to GP increases for higher base-utilization. We can attribute that due to approximation of supply function in the security server. Recall that, the exhaustive search method calculates minimum capacity of the server by exact analysis of the busy period. In contrast, the proposed method approximates the interference to the server from real-time tasks during the interval of server replenishment period and linearize it by taking the ceiling off. While this approximation error is small for low utilization cases, as the base-utilization increases the error accumulates and reduces schedulability. However, still it is possible to accumulate task-sets for higher base-utilization.

The quality of solution (*e.g.,* server utilization) obtained by GP and exhaustive search is illustrated in Fig. 2. The z-axis in this figure represents the difference in server utilization, *e.g.,* $\left(\frac{Q^{\text{EX}}}{P^{\text{EX}}} - \frac{Q^{\text{GP}}}{P^{\text{GP}}} \right)$ where Q^{EX} and Q^{GP} (P^{EX} and P^{GP}) represent the capacity (replenishment period) obtained

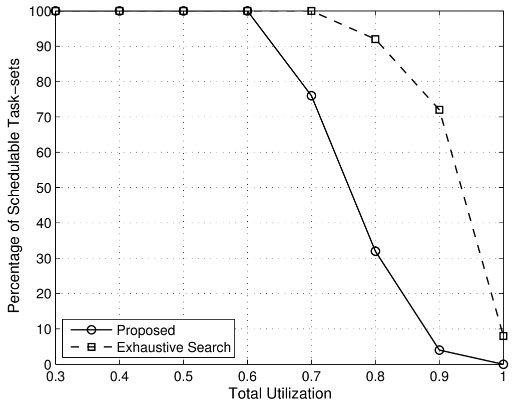


Figure 1. Normalized percentage of the number of schedulable task-sets. The base-utilization of the real-time tasks are varied from $[0.01 + 0.1 \cdot i, 0.1 + 0.1 \cdot i]$ where $0 \leq i \leq 8, i \in \mathbb{Z}$. The utilizations of the security tasks are generated from $[0.11, 0.20]$. For exhaustive search, we set $P_{max} = 2500$ with search granularity $\delta = 0.5$.

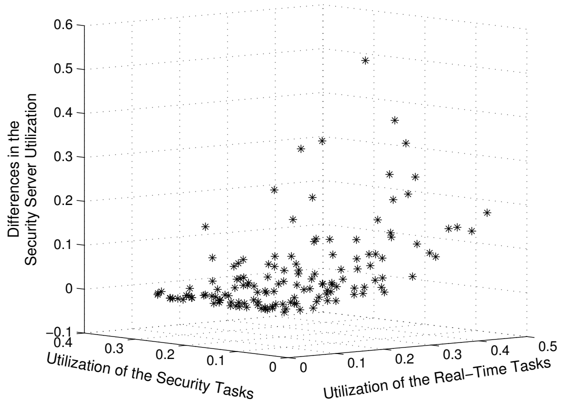


Figure 2. Exhaustive search vs. proposed approach: difference in the server utilization for schedulable task-sets. For each utilization group, we randomly generate 100 task-sets and compare the schedulability of both schemes.

[7]Similar search method has also been discussed in literature [10], [14], [16].

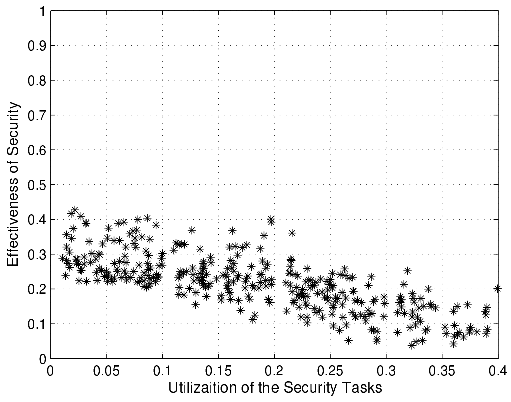


Figure 3. The effectiveness of security is measured as $\xi = \frac{\|\mathbf{T}^* - \mathbf{T}^{\mathbf{des}}\|_2}{\|\mathbf{T}^{\mathbf{max}} - \mathbf{T}^{\mathbf{des}}\|_2}$. The base-utilization of the real-time tasks are taken from $[0.31, 0.4]$ and the base-utilization of the security tasks are varied from $[0.01 + 0.1 \cdot i, 0.1 + 0.1 \cdot i]$ where $0 \leq i \leq 3, i \in \mathbb{Z}$. Hence the total utilization of the system varied from $[0.3, 0.8]$. Each utilization group contains 100 task-sets.

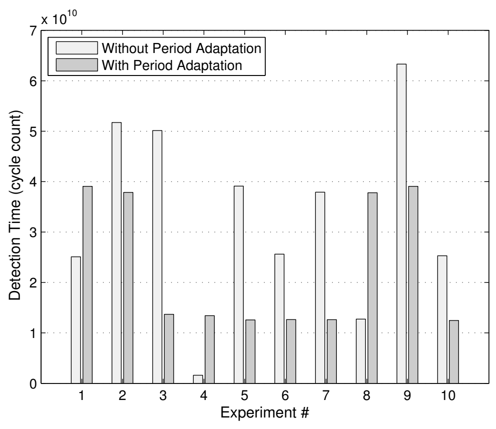


Figure 4. Time to detect attacks: when the periods are optimized vs when set to maximum allowable frequency. Each of the bar-groups represents individual experiment run. We use read time-stamp counter (RDTSC) instruction to measure the cycle count.

from exhaustive search and proposed method, respectively. For low-to-medium utilization cases, the difference is close to zero, which implies the quality of the solution obtained by the GP method is similar to that of obtained by exhaustive search. However, when the utilization is higher the exhaustive search outperforms the proposed method. Again, we can attribute this due to the approximation of supply function in the security server.

It is worth noting that the solution obtained by exhaustive search may not be optimal in a sense that the actual replenishment period may appear beyond P_{max}. As we can see from Fig. 2, for some task-sets the difference is less than zero, *e.g.,* $\frac{Q^{\mathrm{EX}}}{P^{\mathrm{EX}}}$ is lower than $\frac{Q^{\mathrm{GP}}}{P^{\mathrm{GP}}}$. We highlight that the actual search region to find the optimal server parameters for exhaustive search may widely vary based on task-set inputs; and can only be found numerically by trial-and-error. Instead, the proposed method provides a generic approach to analyze the system that is independent of task-set input parameters.

We note that the proposed GP-based approach can solve a given task-set in *seconds*, while the exhaustive search method generally takes few minutes to couple of hours depending on the size of P_{max} and the search granularity δ. Besides, the exhaustive search method is not scalable for task-sets with large number of tasks.

2) Effectiveness of Security: In Fig. 3, we observe the effectiveness of security of the system by means of tightness of the desired periods, *e.g.,* $\xi = \frac{\|\mathbf{T}^* - \mathbf{T}^{\mathbf{des}}\|_2}{\|\mathbf{T}^{\mathbf{max}} - \mathbf{T}^{\mathbf{des}}\|_2}$ 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 to 0, the nearer the period of each of the security task is to the desired period. Each of the data point in Fig. 3 represents schedulable task-sets. We find that most cases the algorithm finds the periods that is within 20% of the desired period value.

We note that, our GP approach took 5.33 second on average with 23.33 standard deviation to analyze a set of tasks with parameters specified by Table III. Thus, the proposed approach solves problems of reasonable size in *seconds*, which is an acceptable amount of time for *offline analysis*.

3) Experiment with a Security Application: With a view to observing that whether the security tasks can perform desired checking after period adaptation, we perform experiments with Tripwire in RTAI-patched Linux RTOS. For this, we purposely compromise one of the tasks and launch the attack by modifying the contents in the `/sbin` directory. For each of the experiments run, we start with a clean system state, launch attack at any random point of the task execution and log the cycle count requires to detect the attack by Tripwire. We obtain the periods by solving the period adaptation problem and set it as the period of Tripwire in the RTOS. For non-adaptive case, period of Tripwire is set to T_i^{max}. Each of the bars in the x-axis of Fig. 4 denotes individual experiment run, and y-axis shows the corresponding cycle counts to detect the attack. As shown in Fig. 4, out of 10 experiments we find 3 instances, where the non-adaptive period assignment outperforms. This is mainly because the actual time to detect the attacks depends on when the attack is launched and the corresponding scheduling point of the security routine. However in general case, the proposed period adaptation mechanism provides better monitoring frequency and hence it is more likely to detect breaches as soon the attack is launched.

VIII. Limitations and Discussion

While the proposed method provides an approach for integrating security in RTS and a metric to measure the effectiveness of such security integration, there are still some areas for improvement for the mechanisms presented in this paper. The proposed server-based approach imposes additional constraints on periodic monitoring compared to non-server case and also based on approximation of supply function in the security server which somewhat limits schedulability. However, as we discuss in the following, the proposed frameworks allows us to provide better control in terms of security enforcement.

Aforesaid, in order to integrate security policy into the system along with monitoring frequency, other performance aspects such as responsiveness and atomicity of the security tasks also need to be considered. While the proposed server-based method provides us a first step to ensure periodic monitoring, this approach can be extended to satisfy responsiveness and atomicity properties as well.

In the case when responsiveness and atomicity of the security mechanism should be ensured, the proposed framework can be modified as follows. Whilst the security task requires better responsiveness and/or needs to perform special atomic operation, the priority of the server can be increased to a priority that is strictly higher than all (or some) of the real-time tasks, depending on the requirements of the security event. Besides, if the security task running under server is not the highest-priority security task, the priority of that task itself is also increased. If the server's capacity is exhausted while executing any atomic operation or fine-grained checking, we allow the server to overrun, *e.g.,* the server continues to execute at the same priority until the security checking is completed. When the server overruns, the allocated capacity at the start of the next server replenishment period is reduced by the amount of the overrun.

It is worth mentioning that, the cost of responsiveness and atomicity by means of priority inversion will be reflected by compromising the timing constraints of some of the real-time tasks. In such cases, the schedulability analysis need to be performed considering maximum blocking time of the security events. Besides, The scheduling policy should identify which real-time or security tasks can be dropped to provide better trade-off between control system performance and defense against security vulnerabilities. In addition, depending on the actual implementations of the security routines, the scheduling framework may need to follow certain precedence constraints. Analogous to the task example illustrated in Table I, in order to ensure that the security application has not been compromised, the security application's own binary may be scanned first before checking the system binary or library files. We intend to explore these aspects in our future work.

IX. Related Work

Despite the fact that malware developers and sophisticated adversaries are able to overcome air-gaps, most RTS were considered to be invulnerable against software security breaches and until recently security issues in RTS were not extensively discussed in industry or academia. The issues regarding information leakage through storage timing channels using architectural resources (*e.g.,* caches) shared between real-time tasks with different security levels is considered in the work [8], [9]. The authors proposed a modification to the fixed-priority scheduling algorithm and introduced a state cleanup mechanism to mitigate information leakage through shared resources. The cost paid for enforcing security mechanism is the reduced schedulability of overall system.

There has been some work [6], [7] on reconciling the addition of security mechanisms into RTS that considered periodic task scheduling where each task requires a security service whose overhead varies according to the quantifiable level of the service. A new scheduler [6] and enhancements to existing EDF scheduler [7] is proposed to meet real-time requirements while maximizing the level of security achieved. In contrast, we consider a fixed-priority scheduling mechanism where security policies are executed sporadically using a server while meeting real-time requirements.

Although not in the context of security in RTS, a similar line of work to ours exists where the authors statically assign the periods for multiple independent control tasks considering control delay as a cost metric [20]. The cost functions are assumed to be linear in task periods. The control delay is estimated using an approximate response-time analysis and the authors presented an iterative procedure, where the actual (*e.g.,* nonlinear) cost functions are linearized around the current solution in each step. In contrast, our goal is to ensure security of the system without violating timing constraints of the real-time tasks. Hence, instead of minimizing response time, our goal is to assign best possible periods, so that the perturbation between achievable period and desired period is minimized for all the security tasks.

The notion of randomization has been used in literature [21] with a view to hardening security mechanisms by minimizing predictability of deterministic RTS schedulers. The authors proposed a schedule obfuscation method that aimed at randomizing the task schedule while providing the necessary real-time guarantees for safe operation. It is not inconceivable that the randomization protocol in the work [21] is complementary to those presented here and can be adopted to the proposed framework in order to make the system robust against attackers. Different from our work at the scheduler level, architectural frameworks *e.g.,* [22], [23] aim to create hardware/software mechanisms to protect against security vulnerabilities. It is worth mentioning that these two sets of approaches could be combined to make the RTS more resilient to attacks.

X. Conclusion

The evidence from recent successful attacks on automobiles [24], industrial control systems [25] and UAVs [26] indicate that RTS are not invulnerable to security breaches. In this work we are stepping towards developing an integrated security-aware RTS and provide a glimpse of security design metrics for RTS. By using approaches such as the ones presented in this paper, designers of RTS are now able to improve their security posture and consequently safety – which is the main goal for such systems. This is also a step towards developing security metrics for the field of systems security in general.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. The material in this paper is based upon work supported in part by the U.S. Office of Naval Research (ONR) under award number N00014-13-1-0707, and by the U.S. National Science Foundation (NSF) under award number NSF CNS 14-23334[8].

References

[1] P. Mantegazza, E. Dozio, and S. Papacharalambous, "RTAI: Real time application interface," *Linux Journal*, vol. 2000, no. 72es, p. 10, 2000.

[2] "Using the RDTSC instruction for performance monitoring," *Tech. rep., Intel Corp.*, p. 22, 1997.

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

[4] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Comp. J.*, vol. 29, no. 5, pp. 390–395, 1986.

[5] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Soft. Eng. J.*, vol. 8, no. 5, pp. 284–292, 1993.

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

[7] 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.

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

[9] R. Pellizzoni, N. Paryab, M.-K. Yoon, S. Bak, S. Mohan, and R. B. Bobba, "A generalized model for preventing information leakage in hard real-time systems," in *IEEE RTAS*, 2015, pp. 271–282.

[8]Disclaimer: Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

[10] R. Davis and A. Burns, "An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems," in *IEEE RTNS*, 2008.

[11] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarchical fixed-priority scheduling," in *IEEE ECRTS*, 2002, pp. 152–160.

[12] 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.

[13] S. Boyd and L. Vandenberghe, *Convex optimization*, 2004.

[14] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proc. of ACM int. conf. on embedded soft.*, 2004, pp. 95–103.

[15] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *IEEE RTSS*, 2003, pp. 2–13.

[16] M.-K. Yoon, J.-E. Kim, R. Bradford, and L. Sha, "Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling," in *Proc. of DATE*, 2013, pp. 1313–1318.

[17] M. Chiang, *Geometric programming for communication systems*, 2005.

[18] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[19] A. Mutapcic, K. Koh, S. Kim, L. Vandenberghe, and S. Boyd, "GGPLAB: a simple Matlab toolbox for geometric programming," 2006.

[20] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *IEEE RTSS*, 2008, pp. 291–300.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] D. P. Shepard, J. A. Bhatti, T. E. Humphreys, and A. A. Fansler, "Evaluation of smart grid and civilian uav vulnerability to gps spoofing attacks," in *Proc. of the ION GNSS Meeting*, vol. 3, 2012.