Optimizing Runtime Security in Real-Time Embedded Systems

Cailani Lemieux-Mack[1], Kevin Leach[1], Ning Zhang[2], Sanjoy Baruah[2], and Bryan C. Ward[1]
[1]Vanderbilt University, Nashville, TN, USA
[2]Washington University in St. Louis, St. Louis, MO, USA

Abstract

As real-time embedded systems (RTES) become more prevalent in society, their security is increasingly important. However, as these systems are often safety or mission critical, it is imperative that security instrumentations do not violate real-time constraints, which could cause improper or even unsafe function. Therefore, there is a strong need to both ensure correct real-time performance and maximize system security. This paper describes how this can be viewed as an optimization problem, and presents two case studies where this optimization is formulated as a Mixed Integer Linear Programming (MILP) problem. The first optimization shows how optimization techniques can be applied to choose defenses given available system slack. The second demonstrates how a specific security instrumentation can be tuned to maximize security subject to an allowable runtime expansion. The paper concludes with a discussion and directions for future work.

1 Introduction

Historically, enterprise and general-purpose systems have been the target of the majority of cyberattacks. In recent years, however, attacks on real-time and embedded systems (RTES) have become increasingly prevalent, as attackers have targeted systems such as IoT devices (*e.g.,* Mirai botnet [2]), automotive applications [28], and industrial control systems (ICS) (*e.g.,* TRITON attack [13]). Heart monitors have become non-functional during procedures due to security counter measures (*i.e.*, virus scan) [16]. In a recent grim milestone, the first death attributed to a ransomware attack occurred at a German hospital, which was forced to turn a patient away after its systems were shutdown [14].The United States Department of Homeland Security issued an advisory for a collection of vulnerabilities called "BadAlloc" reported to affect over 25 different real-time operating systems (RTOSes) used in commercial applications ranging from industrial control to IoT to medical devices [35].

The increased security pressure on RTES necessitates stronger defenses. However, defenses for RTES face design constraints that do not apply to general-purpose systems, including: size, weight, and power (SWAP); limited or impoverished computing platforms; and notably, real-time considerations. Therefore, many of defensive techniques face challenges to adoption in the RTES context. There are always overhead costs associated with using any security strategy, and so in order for a defense to be safely incorporated into an RTES, there must be assurances that it will not violate the constraints of the system.

This gives rise to a number of important problems and opportunities for the application of optimization techniques for RTES. Specifically, how should a system be maximally protected against security threats subject to real-time constraints? In general it is difficult, if not impossible, to precisely quantify security as attackers can exploit a system by violating assumptions not considered in security quantization. Nonetheless, there exist many security metrics that can be useful for determining which defenses are stronger than others, and therefore guide more secure system designs through security-cognizant optimization.

In this paper, we provide a discussion and examples of formulating optimizing security strategies while maintaining real-time constrains as a Mixed Integer Linear Programming (MILP) problem. We first discuss optimizing the choice of defense strategies for each task in a task set (Inter-defense Optimization). We then discuss optimizing a single "tuneable" security strategy on a task-by-task basis so that the system as a whole has the highest level of security possible given a particular defensive strategy (Intra-defense Optimization). We end with a discussion of real-time security optimization and potential future work.

2 Background

In this section, we define our task model and notation, and provide background on runtime security mechanisms and their performance impacts.

2.1 Task Model

We consider a task system $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ of n tasks, scheduled on m processors. Each task τ_i is composed of a (potentially infinite) sequence of jobs, that are sporadically released with a minimum inter-arrival time of T_i. C_i^0 is the worst case execution time (WCET) of τ_i when no security mechanism in place. Each task must complete by its deadline D_i. We assume implicit-deadline tasks, and therefore $D_i = T_i$. The utilization of a task is defined by $u_i = C_i^0/T_i$, and the task-system utilization is $U = \sum_{\tau_i \in \mathcal{T}} u_i$.

2.2 Defenses for RTES

The domain and range of cyber attacks is immense. In this work, we limit our discussion to defenses against control-flow hijacking attacks. These are attacks in which a malicious actor utilizes a memory-corruption vulnerability and attempts to redirect the flow of execution in the targeted system [32].

Defenses against such control-flow hijacking attacks universally work by changing runtime properties of the protected application to incorporate security checks or moving code or data to increase uncertainty and apparent complexity for attackers and increase the costs of their probing efforts, thus preventing successful exploitation of vulnerabilities (*i.e.*, moving-target defenses [22]). Therefore, adopting any of the defenses considered can be modeled as having some security benefit, and some overhead on the execution cost C_i^0 of the protected task.

We note that many other security approaches, such as intrusion and anomaly detection, can be realized as independent tasks, rather than an augmentation to the protected task as we consider. However, these however do not prevent exploitation as runtime defenses do, but instead allow an operator to detect malicious activity and respond accordingly. We refer the interested reader to [17, 18] for a discussion of such techniques.

2.3 Runtime Defenses

Here we review several runtime defenses, how they work, and how they affect real-time performance.

Enforcement-based defenses. Enforcement-based defenses protect against memory-corruption attacks by enforcing some type of policy on program execution. This is usually done by gathering metadata about the program, either at compile or runtime, and then inserting checks into the program to verify its state against this metadata. What metadata is collected and where checks are inserted depends on the policy being enforced. Enforcement-based defenses can be further divided into memory-safety policies or various weaker policies. Memory safety is a concept that considers pointers to be unforgeable capabilities to memory objects [20]. As such, pointers are restricted to point only within the memory object to which they correspond. SoftBound [29] is a memory-safety enforcement-based defense that enforces spatial memory safety by creating metadata for all pointers (code or data) and inserting runtime checks to validate the integrity of pointers. While very strong, SoftBound also has exceptionally high overhead compared to other defenses.

There are also enforcement-based defenses that apply weaker policies than full memory safety, focusing on limiting the use of corrupted code pointers. For example, control-flow integrity (CFI) [1] is a defense that leverages runtime monitors to detect control-flow hijacking attacks. This is done by identifying all of the legal transitions between functions for a given program in a control-flow graph (CFG). If a process attempts to make a transition that is not included in the CFG, the monitors will halt the process [8]. This is a strictly weaker policy than memory safety because it only prevents the overwriting of code pointers, and not for example, data pointers or data values.

Other enforcement-based approaches include code pointer integrity (CPI) [24], Safestack [10], and shadow stacks [9].

Randomization-based defenses. Randomization-based defenses are the other class of runtime defenses and they mitigate attacks by randomizing the code or its environment, thus preventing an attacker from exploiting vulnerabilities by creating uncertainty about important information (*e.g.,* addresses of code regions). Randomization-based defenses are categorized based on their granularity.

ASLR [31] is a defense that is used by default on most modern operating systems. This randomization has the coarsest granularity, shifting the entire executable section in memory by a randomly selected value at load time. Other randomization-based defenses we consider include compiler-assisted code randomization (CCR) [23], which permutes functions or basic blocks within the binary at compile time, and the Multicompiler [21], which inserts random no-operations (no-ops) into the binary at compile time to change the layout and offsets of individual instructions.

2.4 Tuneable Defenses

There are a few new defenses that enable a more fine-grained trade-off between security and performance. One such example is the partial context-sensitive pointer-integrity framework (ParCSPI) [36]. ParCSPI uses a technique called pointer authentication to ensure the integrity of a pointer traversed at runtime, *i.e.*, that the pointer has not been corrupted to point to a malicious target. The tuneable parameter is derived from *context sensitivity*, or where the pointer is being used. A pointer may be valid in one calling context, or location in the code, but invalid in another. Enabling full context sensitivity can be comparatively expensive, even though it provides the strongest security. The ParCSPI framework therefore allows for optimizations to be performed to decide where context sensitivity should be applied to maximize security for a given overhead allowance.

As a simplistic example, consider an application with one code path that is much longer than the others, and represents the WCET. Applying context sensitivity to pointers on that code path will increase the WCET. However, context sensitivity can be applied to pointers on other code paths, and so long as they do not increase the execution time of those paths to longer than the WCET of the longest path, they do not affect the WCET. If some WCET expansion is allowable, shorter paths may enjoy more context sensitivity, while the longer path may allow some, but not complete context sensitivity.

3 Inter-defense Optimization

We next describe a case study of an optimization to determine how to protect each task to optimize defensive coverage while respecting real-time constraints. If a system has low utilization and abundant slack time, perhaps stronger, higher-overhead defenses are preferable to lower-overhead defenses that may not provide as much defensive coverage. Conversely, for a system with high utilization and minimal slack time, only low-overhead defenses may be feasible. This gives rise to an important optimization problem: how can we maximize the defensive coverage of a real-time system while maintaining schedulability? To answer this question, we develop a new optimization framework that maximizes defensive coverage while maintaining schedulability. We then present a case study informed by our empirical results that demonstrates its effectiveness.

Quantifying security as a metric is widely known to be a difficult and often imprecise endeavor [19]. Attackers do not comply with stochastic models, and new attack techniques are constantly being developed that undermine previously held assumptions [34]. Nonetheless, efforts at quantifying security can still be useful towards designing new defenses, and encouraging the adoption of stronger security postures.

QUASAR: In this case study we consider the defensive-coverage metric from the Quantitative Attack Space Analysis and Reasoning (QUASAR) tool [33]. QUASAR is based on an expert-developed model of fine-grained individual capabilities that are required to carry out control-flow hijacking attacks. This model can then be synthesized into a Boolean formula in which any solution represents a valid attack. Defenses constrain what capabilities are available to an attacker, and can be incorporated into the Boolean formula, thereby limiting the number of solutions. While generally the problem of solution counting (#SAT) is NP-hard, QUASAR leverages efficient algorithms that scale easily to the model size. We use QUASAR to determine the defensive coverage provided by individual defenses. Defensive coverage represents the percent of valid attacks that are prevented by a defense.

QUASAR$^{\mathbf{RT}}$. We build upon the results of QUASAR and incorporate the defensive overheads from our previous experimental evaluations [7] to develop a new defensive-optimization framework, QUASAR$^{\mathrm{RT}}$. This optimization uses the defensive overheads from our empirical evaluations, and the defensive coverage of several defenses considered in QUASAR as inputs to a mixed-integer linear program (MILP) that can identify the optimal defensive coverage for a given task system. We describe this MILP after first introducing relevant notation.

We assume Global Earliest Deadline First (EDF) scheduling and consider that schedulability is evaluated via a utilization-based schedulability test. For a uniprocessor system, earliest-deadline first (EDF) is optimal and therefore a task system is schedulable if $U \leq 1$. On a multiprocessor system with m processors, global EDF has been proven soft-real-time optimal in that it provides bounded deadline tardiness, and thus a task system is soft-real-time schedulable if $U \leq m$ [12]. In our experimental evaluations, we assume soft-real-time schedulability, *i.e.*, deadline tardiness is bounded, but note that other utilization-based tests could easily be substituted in the optimization. We note that exploring optimizing defensive coverage in the context of more sophisticated schedulability tests and priority-assignment schemes is an interesting avenue of future work.

We assume a set D of defenses, for which each defense $d_j \in D$ is defined by a defensive coverage c_j derived from QUASAR and a multiplicative defense cost δ_j. Thus, the execution time of τ_i when d_j is applied is $C_i^0 \delta_j$. We assume that δ_j is a static value for all tasks. In practice you could have task-specific values, but that is beyond the scope of this paper.

QUASAR$^{\mathbf{RT}}$ MILP formulation. All task and defense parameters in the QUASAR$^{\mathrm{RT}}$ MILP are *constants* with respect to the MILP. The only variables are binary indicator variables that encode whether a defense is applied to a task. These variables are encoded as $x_{i,j}$ for $\tau_i \in \mathcal{T}$ and $d_j \in D$. Consequently, the QUASAR$^{\mathrm{RT}}$ MILP is formulated as follows:

$$\begin{aligned}
\text{Maximize:} \quad & \sum_{\tau_i \in \mathcal{T}} \sum_{d_j \in D} x_{i,j} c_j \\
\text{Subject to:} \quad & \sum_{d_j \in D} \sum_{\tau_i \in \mathcal{T}} C_i^0 \delta_j x_{i,j} \leq m \\
& \sum_{d_j \in D} x_{i,j} \leq 1 && \forall \tau_i \in \mathcal{T} \\
& C_i^0 \delta_j x_{i,j} \leq T_i && \forall \tau_i \in \mathcal{T}, \forall d_j \in D
\end{aligned}$$

The objective function of this MILP is the sum of the defensive coverage for all enabled defenses for each task. The first constraint set encodes that after defenses are applied, the total utilization of the task system does not exceed the total platform utilization.[1] The second constraint set encodes that at most one defense can be applied to each task. This assumption is based on the observation that in general, defenses do not necessarily compose with one another. For example, in the case study we present below, we consider address-randomization defenses of differing granularities, which do not easily compose, for example, because they are alternative compiler tools. Furthermore, the overheads of the composition of defenses have not been evaluated. Relaxing this condition is fertile ground for future research. Finally, the last constraint set encodes that the execution cost of each task after including defensive overhead should not exceed the deadline, otherwise the task itself would not be schedulable.

QUASAR$^{\mathbf{RT}}$ case study. To demonstrate QUASAR$^{\text{RT}}$, we conducted a schedulability study. We considered $m \in \{2, 4\}$, with task systems with total utilization $U \in \{0.05, 0.1, \ldots, 4.0\}$. All per-task utilizations and periods were generated using distributions documented in SchedCAT [6] and used in prior work (*e.g.,* [5]). This resulted in 108 schedulability graphs.

We compare the QUASAR$^{\text{RT}}$ defensive plan against three randomization defenses: ASLR, CCR, and the Multicompiler (MC), as well as an undefended system. (Selfrando [11], and several other enforcement-based defenses considered are not currently modeled in QUASAR [33], but could be.) For each considered defense, we assume the defense was applied to all tasks. For the purpose of computing the average defensive coverage, we assume if a task system is unschedulable, it has zero defensive coverage. This case study demonstrates how QUASAR$^{\text{RT}}$ can be used to determine the best granularity of randomization to be applied across different tasks.

An example schedulability and defensive-coverage graph is depicted in Fig. 1. In the top inset, soft

[1]Alternative utilization-based schedulability conditions could be easily substituted here.

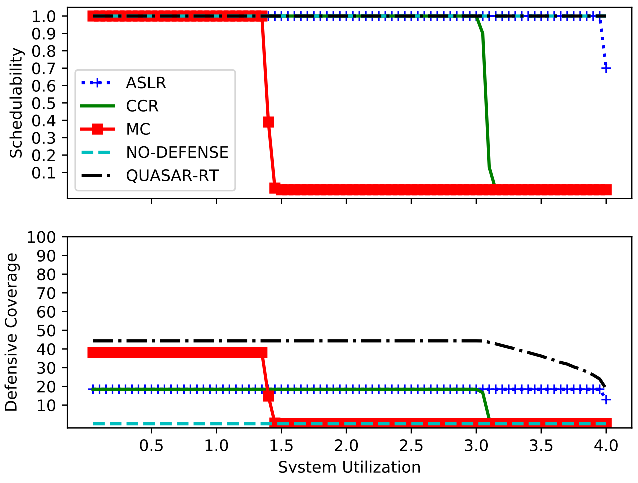

Figure 1: Schedulability graph evaluating randomization defenses.

real-time schedulability is depicted, and in the corresponding bottom inset, the average defensive coverage of the generated task systems is plotted. Observe that QUASAR$^{\text{RT}}$ has the same schedulability as no defenses. This is because the optimization will disable all defenses if necessary to ensure schedulability. This is opposed to defending all tasks with a given defense, where overheads cause varying degrees of utilization loss. Observe that the defensive coverage begins to decline rapidly when the task system is not schedulable using CCR. For these higher-utilization task systems, QUASAR$^{\text{RT}}$ defends as many tasks as possible with the stronger CCR, while defending the remaining with the weaker ASLR or no defense at all. As a result, QUASAR$^{\text{RT}}$ dominates both schedulability and defensive coverage of approaches that universally apply a single defense, or no defense at all.

4 Intra-defense Optimization

Next, we review another case study of optimization to balance security with runtime constraints [36].[2] Given a tunable defense strategy such as ParCSPI, we wish to choose an optimal policy for each task in a task set in order to maximize the total security score without violating schedulability. Thus, we designed a Mixed Integer Linear Programming (MILP) optimization problem that allows us to maximize the security while guaranteeing that the task system remains schedulable.

Assume a trade-off table for each task τ_i has been generated using analysis methods described in [36]. The trade-off table entries represent security score and

[2]This section is adapted from [36], but included to provide another case study in security optimization.

WCET expansion pairs. Each row k in the table contains an overhead ϵ_{ik} and security score S_{ik} pair for some security policy π_{ik} for τ_i. Let the number of entries in this table be h_i. We will call the cost of task $\tau_i \in \mathcal{T}$ with the overhead of the selected security policy C_i.

MILP Formulation. In order to represent choosing a policy, a boolean variable $\mathbf{X}_{ik}$ is defined for each π_{ik}. $\mathbf{X}_{ik} = 1$ implies that τ_i is using π_{ik} and so achieving a security score of S_{ik} and incurring an overhead of ϵ_{ik}.

Thus, the optimization criterion is as follows:

$$\text{maximize} \sum_{i=1}^{n} \sum_{k=1}^{h_i} \mathbf{X}_{ik} S_{ik} \tag{1}$$

A task can only use one policy at a time. Therefore the following constraint is introduced:

$$\forall \tau_i \in \mathcal{T} : \sum_{k=1}^{h_i} \mathbf{X}_{ik} \leq 1 \tag{2}$$

This enforces that at most one $\mathbf{X}_{ik}$ is set to 1.

In order to require that all tasks must remain schedulable, schedulability can be represented as an Integer Linear Program as described by Baruah & Ekberg [3].

They assume deadline-monotonic (DM) scheduling on a single processor and that therefore, the task set has been sorted by relative deadlines so that the priority of τ_i is greater than the priority of τ_{i+1}.

The ILP representation of response-time analysis in [3] proceeds as follows:

For each $\tau_i \in \mathcal{T}$ a new variable $\mathbf{R}_i$ is defined that represents the response time of τ_i. In order for the task system to remain schedulable, each task must meet its deadline D_i, so a new constraint is introduced:

$$\forall \tau_i \in \mathcal{T} : \mathbf{R}_i \leq D_i \tag{3}$$

They introduce a new variable to represent the $\lceil \mathbf{R}_i / T_j \rceil$ term from standard response time analysis. For each task τ_i and each higher-priority task τ_j, they create a new non-negative integer variable $\mathbf{Z}_{ij}$ that represents an upper bound on the number of times that τ_j can run during the response time of τ_i. Thus, a new constraint is defined:

$$\mathbf{Z}_{ij} \geq \left(\frac{\mathbf{R}_i}{T_j} \right) \tag{4}$$

Finally, in order to represent the mathematical model of schedulability Baruah and Ekberg use the following constraint:

$$\forall \tau_i \in \mathcal{T} : C_i + \sum_{j=1}^{i-1} \mathbf{Z}_{ij} C_j \leq \mathbf{R}_i \tag{5}$$

However, this assumes that the cost for each task is a constant. We violate that assumption since the overhead of each possible policy is different. Thus, it is necessary to differentiate between the original cost of τ_i which here is notated as C_i^0 and the actual cost of τ_i which is C_i. [36] defines C_i in terms of C_i^0 as:

$$C_i = C_i^0 \left(1 + \sum_{k=1}^{h_i} (\mathbf{X}_{ik} \epsilon_{ik}) \right) \tag{6}$$

Note that since only one X_i can be true at any time, and each X_i is a binary variable, the original cost will only be altered by the chosen ϵ_i.

For ease of presentation, they define an intermediate variable H_i that represents the effect of higher priority tasks on the response time of τ_i. In Constraint (5), this is equivalent to $\sum_{j=1}^{i-1} \mathbf{Z}_{ij} \times C_j$, however given the definition of C in terms of C^0, it can be expressed as:

$$H_i = \sum_{j=1}^{i-1} [(\mathbf{Z}_{ij} C_j^0) + C_j^0 \sum_{k=1}^{h_j} (\mathbf{Z}_{ij} \times \mathbf{X}_{jk} \times \epsilon_{jk})] \tag{7}$$

Note that $\mathbf{Z}_{ij}$ has been distributed into the summation. Importantly, it can be seen that there is the multiplication of two variables $\mathbf{Z}_{ij}\mathbf{X}_{jk}$ making the constraint seemingly non-linear.

[36] gets around this problem by leveraging the observation that the result of the multiplication is either $\mathbf{Z}_{ij}$ or 0 since X_{jk} is a boolean. Thus, they can use well-known integer-programming techniques for linearizing products. They define new variables $\mathbf{Y}_{ijk}$ to represent the product $\mathbf{Z}_{ij}\mathbf{X}_{jk}$. Additionally, they define a constant $Z_{ij}^U = \lceil T_i / C_j^0 \rceil$ as an upper bound for $\mathbf{Z}_{ij}$. Since it is required that everything remains schedulable, and τ_j will not run more often than its period. They can upper bound $\mathbf{Z}_{ij}$ with the number of times that C_j^0 could fit into T_i. The following constraints are defined for each $\mathbf{Y}_{ijk}$:

$$\begin{aligned} 0 &\leq \mathbf{Y}_{ijk} \\ \mathbf{Y}_{ijk} &\leq \mathbf{X}_{jk} Z_{ij}^U \\ 0 &\leq \mathbf{Z}_{ij} - \mathbf{Y}_{ijk} \\ \mathbf{Z}_{ij} - \mathbf{Y}_{ijk} &\leq (1 - \mathbf{X}_{jk}) Z_{ij}^U \end{aligned} \tag{8}$$

The first two inequalities ensure that $\mathbf{Y}_{ijk} = 0$ when $\mathbf{X}_{jk} = 0$. The second two inequalities ensure that

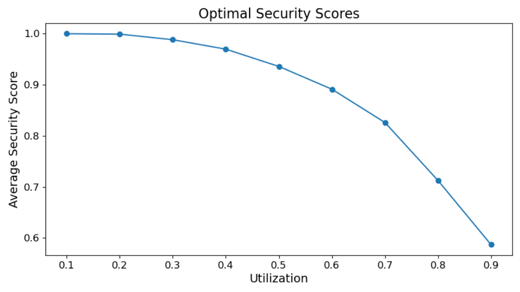

Figure 2: Optimal average security scores per task vs total utilization of synthetic task sets.

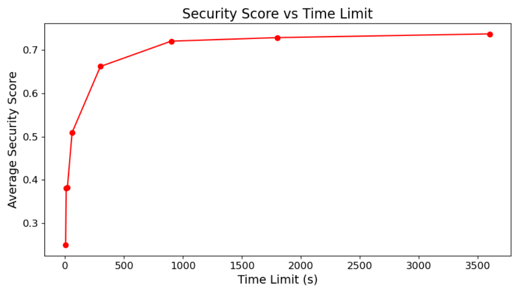

Figure 3: Average security score found over time.

$\mathbf{Y}_{ijk} = \mathbf{Z}_{ij}$ when $\mathbf{X}_{jk} = 1$. They can now use $\mathbf{Y}_{ijk}$ to update the definition of H_i so that it is linear:

$$H_i = \sum_{j=1}^{i-1} \left((\mathbf{Z}_{ij} C_j^0) + C_j^0 \sum_{k=1}^{h_j} (\mathbf{Y}_{ijk} \times \epsilon_{jk}) \right) \tag{9}$$

Finally, the defined schedulability constraint is:

$$\forall \tau_i \in \mathcal{T} : C_i + H_i \leq R_i \tag{10}$$

Subject to the defined constraints, the defined MILP optimization problem can choose policies that maximize the sum of the security scores for each task while guaranteeing that the task set remains schedulable.

4.1 Case-Study Evaluation

In order to evaluate this case study, we present two experiments. The first experiment aims to determine the optimal security level of synthetic task sets given different original system-utilization levels. The second experiment evaluates the trade-off between solver performance vs. execution time. These experiments are run over a randomly generated synthetic task set assigning each task real world security vs. overhead trade-offs determined from the BEEBS dataset [30].

The trade-offs are represented in a "trade-off table" for each task. Schedcat TaskGenerator is used for the synthetic task-set generation [6]. The optimization used the GNU Linear Programming Kit (GLPK) solver using the PuLP API on an 8 Core, 3.0GHz CPU with 16 GiB Memory.

For the optimal security level experiment, the solver is run for each system-utilization level under test. Total utilizations in $\{0.1, 0.2, \ldots, 0.9\}$ are considered. Task periods are generated by sampling uniformly among $[10, 100]$ms. This is equivalent to the built-in 'uni-moderate' distribution as used in [5, 6]. For the task utilizations, uniformly random choices from $[0.1, 0.4]$ are used. This is the built-in 'uni-medium' distribution.

Over 3000 task sets per utilization level were generated and any task sets that were not schedulable before instrumentation were discarded. We randomly assigned a trade-off table to each task in the task set from the tables measured from the BEEBS benchmark. Finally, the optimization was performed on each task set using a time limit of 1 hour per optimization. The security scores per task were averaged for each generated task set and then averaged per generated system utilization value. A graph of results of this is in Figure 2.

We observe the average per-task security score decreases as the original system utilization increases. This is because in order to achieve higher security there is more overhead. Thus, when the original utilization is higher and there is less available time, it is not possible to achieve as high a security score.

MILP is a known NP-complete problem, so it is important to evaluate its scalability. [36] found that for task systems with more than approximately 10 tasks, the execution time was often over one hour. However, MILP solvers can often quickly find *feasible* solutions, and much of the execution time can often be spent finding the optimal value. Therefore, the solution quality is evaluated over time that the solver has run. The solutions given by the solver are compared after 5 seconds, 10 seconds, 20 seconds, 1 minute, 5 minutes, 15 minutes, 30 minutes, and 1 hour.

For this experiment five unique task sets are generated. The maximum utilization is defined to be 0.6. In this experiment the Schedcat 'uni-light' distribution, which is drawn uniformly from $[0.001, 0.1]$ is used for the period in order to increase the number of generated tasks. These settings are chosen to make the optimization more difficult and to force the number of tasks in the task set to be higher and thus to make the optimization more difficult. Additionally the tasks are limited to using a singe trade-off table for consistency. Specifically, the tasks use the table for the BEEBS `st` program. The solver was run for each task set and time limit and the per-task average security score was collected. The results of this experiment can be found in

Figure 3.

We observe that while one hour may not be long enough to definitively determine the optimal solution, the average security score converges quickly, demonstrating that while the optimal solution may not be easily found, this formulation can be practically applied to improve the security posture of the system, even for larger task systems.

5 Discussion and Related Work

The optimization discussed in Section 4 can be easily extended to support multiple partitioned processors. Starting from the multi-processor MILP response time analysis presented by Ekberg and Baruah [15], a similar derivation to Section 4 could be applied to trivially extend the technique to partition-based multi-processor scheduling. We note, however, that many other results from Ekberg and Baruah [15] suggest that there is not likely to be an ILP formulation for other schedulers, or task systems with constrained or arbitrary deadlines. For example, they showed that EDF scheduling of sporadic tasks with constrained deadlines is both strongly NP-hard and coNP-hard, and therefore unlikely to be in either class, and thus likely cannot be formulated as an ILP in polynomial time.

There has been other recent work on optimizing security subject to real-time constraints. Di Leonardi et al. [25] presented a optimization problem that allows different combinations of security techniques to be applied for each basic block in a task set while maintaining system-wide schedulability. It is both an inter-defense and intra-defense optimization technique as it chooses between different security techniques for each block as well as determines which basic blocks a given security technique is best applied within a single task. However, the optimization can only work with techniques that both apply only on the basic-block level and can safely be composed with other technique. We note that defense composition can be challenging both in practical terms, as different defenses can be implemented in different toolchains, and theoretically, as in some cases different defenses have different assumptions or limitations, and mixing them can invalidate such assumptions. Cross-language attacks are one such example [27].

Lin et al. [26] presented an inter-defense optimization formulation that assigns groups of security services to best fulfil the security requirements of a task system. Optimization techniques have also been applied to minimize the worst-case performance effects of a security protection called data-flow integrity (DFI) [4], but that work was focused on optimizing the protection of an individual task. Interesting future work could investigate incorporating similar techniques into a system-wide optimization, and/or considering it in conjunction with other defensive techniques.

6 Conclusion

Real-time embedded systems often require strong security guarantees. However, security defenses can break real-time constraints if their effects are not accounted for. We provide a discussion of optimization for security in RTES. We show two case studies of MILP formulations to maximize security while maintaining schedulability.

References

[1] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 340–353. ACM, 2005.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110. USENIX Association, August 2017.

[3] Sanjoy Baruah and Pontus Ekberg. An ILP representation of Response Time Analysis. Available at `https://research.engineering.wustl.edu/~baruah/Submitted/2021-ILP-RTA.pdf`, 2021.

[4] Nicolas Bellec, Guillaume Hiet, Simon Rokicki, Frederic Tronel, and Isabelle Puaut. RT-DFI: Optimizing data-flow integrity for real-time systems. In *34th Euromicro Conference on Real-Time Systems, ECRTS 2022*, number 34, 2022.

[5] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, Chapel Hill, NC, August 2011.

[6] B. Brandenburg. SchedCAT: The real-time scheduling toolkit. `https://github.com/brandenburg/schedcat`, 2024. Accessed: 2024-10-22.

[7] Nathan Burow, Ryan Burrow, Roger Khazan, Howard Shrobe, and Bryan C. Ward. Moving target defense considerations in real-time safety- and mission-critical systems. In *Proceedings of the 7th ACM Workshop on Moving Target Defense*, MTD'20, 2020.

[8] Nathan Burow, Scott A. Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. Control-flow integrity: Precision, security, and performance. *ACM Comput. Surv.*, 50(1), April 2017.

[9] Nathan Burow, Xinping Zhang, and Mathias Payer. SoK: Shining light on shadow stacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 985–999. IEEE, 2019.

[10] Gang Chen, Hai Jin, Deqing Zou, Bing Bing Zhou, Zhenkai Liang, Weide Zheng, and Xuanhua Shi. SafeStack: Automatically patching stack-based buffer overflow vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 10(6):368–379, 2013.

[11] Mauro Conti, Stephen Crane, Tommaso Frassetto, Andrei Homescu, Georg Koppen, Per Larsen, Christopher Liebchen, Mike Perry, and Ahmad-Reza Sadeghi. Selfrando: Securing the tor browser against de-anonymization exploits. *Proceedings on Privacy Enhancing Technologies*, 2016(4):454–469, 2016.

[12] UmaMaheswari C. Devi and James H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.

[13] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. TRITON: The first ICS cyber attack on safety instrument systems. In *Proc. Black Hat USA*, pages 1–26, 2018.

[14] Melissa Eddy and Nicole Perloth. Cyber attack suspected in German woman's death. `https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomeware-death.html`.

[15] Pontus Ekberg and Sanjoy Baruah. Partitioned scheduling of recurrent real-time tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 356–367, 2021.

[16] Dan Goodin. That time a patient's heart procedure was interrupted by a virus scan. May 2016. `https://arstechnica.com/information-technology/2016/05/faulty-av-scan-disrupts-patients-heart-procedure-when-monitor-goes-black/`.

[17] Monowar Hasan. *Integrating security into real-time cyber-physical systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2020.

[18] Monowar Hasan, Ashish Kashinath, Chien-Ying Chen, and Sibin Mohan. SoK: Security in real-time systems. 56(9), 2024.

[19] Cormac Herley and Paul C Van Oorschot. SoK: Science, security and the elusive goal of security as a scientific pursuit. In *2017 IEEE symposium on security and privacy (SP)*, pages 99–120. IEEE, 2017.

[20] Michael Hicks. What is memory safety? `http://www.pl-enthusiast.net/2014/07/21/memory-safety/`, 2021.

[21] Todd Jackson, Babak Salamat, Andrei Homescu, Karthikeyan Manivannan, Gregor Wagner, Andreas Gal, Stefan Brunthaler, Christian Wimmer, and Michael Franz. Compiler-generated software diversity. In *Moving Target Defense*, pages 77–98. Springer, 2011.

[22] Sushil Jajodia. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science+ Business Media, 2011.

[23] Hyungjoon Koo, Yaohui Chen, Long Lu, Vasileios P Kemerlis, and Michalis Polychronakis. Compiler-assisted code randomization. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 461–477. IEEE, 2018.

[24] Volodymyr Kuznetsov, László Szekeres, Mathias Payer, George Candea, R Sekar, and Dawn Song. Code-pointer integrity. In *Proceedings of USENIX OSDI*, 2014.

[25] Sandro Di Leonardi, Federico Aromolo, Pietro Fara, Gabriele Serra, Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Maximizing the security level of real-time software while preserving temporal constraints. *IEEE Access*, 11:35591–35607, 2023.

[26] Man Lin, Li Xu, Laurence T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. Static security optimization for real-time systems. *IEEE Transactions on Industrial Informatics*, 5(1):22–37, 2009.

[27] Samuel Mergendahl, Nathan Burow, and Hamed Okhravi. Cross-language attacks. In *NDSS*, 2022.

[28] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[29] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. SoftBound: Highly compatible and complete spatial memory safety for C. In *ACM Conference on Programming Language Design and Implementation*, PLDI, 2009.

[30] James Pallister, Simon Hollis, and Jeremy Bennett. BEEBS: Open benchmarks for energy measurements on embedded platforms. *arXiv preprint arXiv:1308.5174*, 2013.

[31] PaX. Pax address space layout randomization, 2003. `http://pax.grsecurity.net/docs/aslr.txt`.

[32] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *ACM Conference on Computer and Communications Security*, CCS, 2007.

[33] Richard Skowyra, Steven R Gomez, David Bigelow, James Landry, and Hamed Okhravi. QUASAR: Quantitative attack space analysis and reasoning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 68–78, 2017.

[34] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *Proc. of IEEE Symposium on Security and Privacy*, 2013.

[35] United States Department of Homeland Security. ICS advisory (ICSA-21-119-04). `https://us-cert.cisa.gov/ics/advisories/icsa-21-119-04`.

[36] Yuljie Wang, Cailani Lemieux-Mack, Thidapat Chantem, Sanjoy Baruah, Ning Zhang, and Bryan C. Ward. Partial context-sensitive pointer integrity for real-time embedded systems. In *45th IEEE Real-Time Systems Symposium*, 2024.