Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

Maximizing the Security Level of Real-Time Software while Preserving Temporal Constraints

SANDRO DI LEONARDI[1], FEDERICO AROMOLO[1], PIETRO FARA[1], GABRIELE SERRA[1], (Member, IEEE), DANIEL CASINI[1], (Member, IEEE), ALESSANDRO BIONDI[1], (Member, IEEE), GIORGIO BUTTAZZO[1], (Fellow, IEEE)

[1]Scuola Superiore Sant'Anna, Pisa, Italy

Corresponding author: S. Di Leonardi (e-mail: sandro.dileonardi@santannapisa.it).

ABSTRACT Embedded computing systems are becoming increasingly relevant in the Internet of Things (IoT) and edge computing domains, where they are often employed as the control entity of a cyber-physical system. When operating in such interconnected domains, a software system is susceptible to cyber-attacks from external agents, which can compromise the correct behavior of the system. In addition, the software executing in these systems is typically characterized by stringent timing constraints, which must be satisfied during system execution. Enabling software protections to enhance the security level of the embedded software comes at the cost of increasing the computation times of the tasks, introducing the risk of deadline misses that could also jeopardize the system behavior. This paper presents a methodology to optimize the security level of real-time software while preserving system-wide schedulability by leveraging timing analysis. The proposed approach is based on a mixed-integer linear programming (MILP) formulation that maximizes the security level of the tasks and integrates a response-time analysis technique to assess the schedulability of the system whenever additional protections are activated to shield the software against cyber-attacks targeting specific classes of vulnerabilities. An experimental evaluation is presented to assess the performance of the proposed approach on a representative set of tasks included in standard benchmarking suites for embedded software.

INDEX TERMS Real-time systems, schedulability analysis, cyber-security, vulnerability, optimization.

I. INTRODUCTION

In the domain of cyber-physical systems, real-time embedded computing systems are typically employed to control and monitor a physical process. Cyber-physical systems most often feature an external interface to perform distributed process control, real-time monitoring and data collection, and high-level supervision tasks. When operating in interconnected domains of this kind, an embedded software system is open to cyber-attacks that can be carried out by external agents, introducing additional risks concerning the security of the cyber-physical system. Compromising the security of such a software system constitutes an immediate and critical threat to the safe operation of the physical process, which must be carried out in accordance with strict safety criteria and operational constraints to guarantee both performance and safety. The design of real-time embedded software requires coping with timing constraints, which mandate that each computational task, typically executed in a periodic or sporadic fashion, is assigned a completion deadline corresponding to the time by which the task should finish its job with respect to its release time. Safety-critical systems typically include a subset of hard real-time tasks whose deadlines should be guaranteed off-line in all execution scenarios to ensure the correct operation of the system. This is relevant in several application domains, such as robotics [1]–[4], autonomous driving [5]–[8], smart cities [9, 10], and Industry 4.0 [11, 12].

The implementation of software security mechanisms to protect a program against a set of known cyber-attacks [13] comes at the cost of introducing non-negligible overheads in the execution of the targeted software systems [14]. When a large number of software protections are activated for a given program, the resulting processing overheads may hence significantly affect the timing performance of the system due

to, for instance, the increase of the execution times of the tasks. This may clearly jeopardize the satisfaction of timing constraints.

It is hence relevant to study how to best protect real-time systems from cyber-attacks while still guaranteeing deadlines. This requires dealing with multiple challenges, such as **(i)** determining appropriate parameters to model the security of a real-time application, **(ii)** including them in a new, security-aware, real-time task model, **(iii)** selecting suitable analysis methods to guarantee the real-time constraints of target applications, and **(iv)** devising optimization strategies to maximize security while guaranteeing the temporal constraints.

Contribution. This work presents a methodology to optimize the security level of real-time software while preserving its schedulability. In particular, it provides the following contributions:

- A security-aware real-time task model that allows describing the internal structure of each task by means of its control-flow graph while capturing the security vulnerability that can affect each individual basic block (sequential blocks of execution within the code) and the corresponding impact of installing security protections.
- A *mixed-integer linear programming* (MILP) formulation that allows maximizing the security level of a given task set by selectively activating software protections for each task while ensuring that the task set remains schedulable.
- An extensive experimental evaluation to assess the performance and efficiency of the proposed approach based on representative task sets taken from state-of-the-art benchmarking suites for embedded software.

The optimization is carried out at the level of the basic blocks of each task. To verify system schedulability, an efficient response-time analysis technique for fixed-priority scheduling is integrated into the MILP formulation.

To the best of our knowledge, no prior work is specialized on deriving an optimal trade-off of this kind, instead focusing on either selectively applying specific protections or analyzing real-time software with pre-configured protections in terms of its timing properties.

Paper structure. The rest of this paper is organized as follows. Section II introduces the system model and the terminology used in the paper and identifies the problem approached by the proposed methodology. Section III introduces a characterization of relevant classes of software vulnerabilities and the related protections considered in the paper. Section IV provides an overview of the proposed analysis and optimization approach, while Section V presents the proposed MILP formulation. Section VI provides the results of an experimental evaluation of the proposed approach. Section VII presents an overview of related works in the literature. Finally, Section VIII concludes the paper.

II. SYSTEM MODEL AND PROBLEM STATEMENT

This section first provides an overview of the proposed methodology to perform a joint security- and timing-related optimization starting from the application source code. Then, it introduces the task model, as well as the security and threat model.

A. OVERVIEW

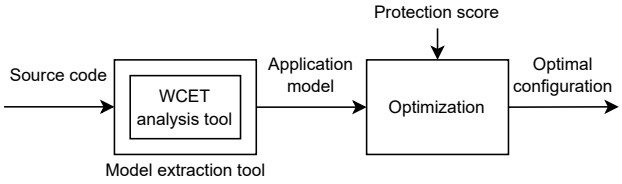

FIGURE 1. Workflow of the proposed approach.

This paper aims at answering the following question:

Research question. *Starting from the source code of an application composed of multiple real-time tasks, how to derive the optimal trade-off between the security protections to be enabled in the application and its schedulability?*

Clearly, to maximize security, one should implement as many security protections as possible. On the other hand, enabling security protection mechanisms introduces non-negligible overheads that can easily jeopardize schedulability. Therefore, a potential trade-off arises between security and schedulability. The exploration of this trade-off can be carried out by means of optimization methods, provided that suitable models for both the application and the corresponding security protections are available. Differently from prior work (see Section VII for a detailed discussion), we tackle this optimization problem at a fine-grain level. In particular, for each software task in the system, we leverage a *control-flow graph* (CFG) that encodes all the possible execution paths of the task. The CFG nodes correspond to the *basic blocks* (BBs) of the program, i.e., sequential blocks of execution, for which the optimizer is called to decide whether it is convenient to enable a set of security protection mechanisms (if any), also considering that different protections applied to different BBs do not contribute the same to the overall security level of the system.

To this end, we propose the following workflow, illustrated in Figure 1. First, a model extraction tool is used to extract the CFG and the corresponding *worst-case execution time* information for each task in the target application. This procedure, detailed in Section IV, leverages the *Heptane* state-of-the-art static WCET analysis tool [15]. The same tool is also adopted to quantify the overhead introduced by enabling each security protection at the level of the BBs.

The resulting information is then used to generate an optimization problem based on the execution paths in the CFGs that can be encountered by executing the tasks. The optimization problem is based on an MILP formulation,

presented in Section V, which allows deriving the most convenient security protections to be enabled to maximize the security level of the application while guaranteeing its schedulability. Protection scores to quantify the security level are derived according to the relevance of well-known vulnerabilities, leveraging information obtained from cyber-security vulnerability databases, as discussed in Section III.

B. TASK MODEL

This work considers an application consisting of a set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of n sporadic real-time tasks to be executed on a multiprocessor platform consisting of m identical processing cores (or processors) $P_1, \ldots, P_m$. Tasks are managed according to *partitioned fixed-priority scheduling*, meaning that each task is assigned to a core at design time, and, at runtime, the tasks assigned to each core are managed by a fixed-priority scheduler for uniprocessors. This scheduling strategy is highly predictable from a timing perspective [16, 17] and is available in many popular target operating systems. For example, on Linux, partitioned fixed-priority scheduling can be configured by using the `SCHED_FIFO` scheduling class and specifying affinities to individual cores with the `sched_setaffinity()` system call. The processor to which a task τ_i is assigned is denoted by $P(\tau_i)$. Without loss of generality, we assume that each task is assigned a unique priority level. Given an arbitrary task $\tau_i \in \Gamma$ allocated to core P_k, the set of tasks with higher priority than τ_i allocated on the same core is denoted by $\text{hp}_k(\tau_i)$. Each task τ_i releases a potentially infinite sequence of jobs, each separated by the previous one by a minimum inter-arrival time T_i, and is subject to a relative deadline D_i, meaning that each job is expected to terminate within D_i units of time from its release. In the following, we consider the case of constrained deadlines, meaning that the value of the deadline parameter D_i is constrained such that $D_i \leq T_i$.

The computational structure of each task is represented by a *control-flow graph* (CFG) that encodes all the possible execution paths. In particular, each task is composed of a set of $n_{i,\text{B}}$ *basic blocks* (BBs) $B_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,n_{i,\text{B}}}\}$, where each basic block $b_{i,j}$ represents a sequential block of execution within the program. Then, the CFG of a task τ_i consists in a directed graph $G_i = (B_i, E_i)$, where the set of basic blocks B_i corresponds to nodes (or vertices) in the CFG and E_i is a set of edges. Edges in E_i encode the possible sequences of execution of BBs yielded by executing the program, in the sense that an edge between two nodes $b_{i,j}$ and $b_{i,k}$ represents the potential flow of control from the starting BB $b_{i,j}$ towards the destination BB $b_{i,k}$, resulting from the execution of a flow control statement at the end of BB $b_{i,j}$. Such flow control statements include conditional or unconditional branches and function calls. Without loss of generality, it is assumed that each DAG G_i has a single source BB (at the task entry point) and a single sink BB (at the task exit point). Within a CFG G_i of a task τ_i, a *path* is defined as an ordered sequence of BBs from the source BB to the sink BB, where each pair $(b_{i,j}, b_{i,k})$ of adjacent BBs in the path is connected by an edge in E_i, directed from $b_{i,j}$ to $b_{i,k}$.

In order to enable static WCET analysis techniques applicable, an upper bound has to be provided on the number of times in which each cycle in the CFG is traversed. For instance, this information can be provided by means of code annotations [18]. Under this assumption, it is possible to apply standard static program analysis techniques to unroll all cycles that are present in the CFG. The result is a set of paths of finite length for the CFG G_i of a task τ_i, denoted by $\text{paths}(\tau_i)$, where each path in $\text{paths}(\tau_i)$ may traverse each BB in B_i multiple times. In the following, for any path $\lambda \in \text{paths}(\tau_i)$, the notation $b_{i,j} \in \lambda$ is used to indicate that the BB $b_{i,j}$ belongs to the path λ.

An example of CFG representing the control flow of a task is provided in Figure 2(a). The figure highlights a loop in the control flow provided with a bound on the number of times the BBs within the loop are traversed during task execution.

An example intermediate representation of the control flow of the task following a loop unrolling procedure is provided in Figure 2(b). Observe that this representation allows enumerating the paths of the CFG and derive the set of paths $\text{paths}(\tau_i)$.

Each task τ_i is then characterized by a *baseline worst-case execution time* (BWCET) C_i, which denotes the worst-case execution time of the task when no security protection is installed. Then, for each task τ_i, each BB $b_{i,j} \in B_i$ is itself characterized by a BWCET, denoted by $C_{i,j}$, representing the worst-case execution time required to execute the BB when no protection is installed in it. Both parameters can be extracted by state-of-the-art WCET analysis tools [15], as discussed next in Section IV. The response time of a job of a task τ_i is defined as the difference between the finishing time of the job, i.e., the time at which the job completes its execution, and its release time. Then, the *worst-case response time* (WCRT) R_i of the task τ_i is defined as the maximum response time across all possible jobs of τ_i in all possible schedules of the application Γ.

C. SECURITY AND THREAT MODEL

The application Γ may be susceptible to cyber-attacks due to the presence of vulnerabilities in the program executed by each task. For the purpose of deriving a method to optimize the security of the application, we consider vulnerabilities to be classified into types. Several classifications of program vulnerabilities are available in the literature [13, 19]. The set of vulnerability types is denoted by $\mathcal{V}$. For each vulnerability type $v \in \mathcal{V}$, we assume that a corresponding set of security countermeasures is available to shield the program against attacks leveraging that vulnerability. The countermeasures available for each vulnerability are modeled as a protection mechanism for the specific vulnerability v, which can be installed within the program at the level of BBs.

Parameter vi, j is introduced for each BB $b_{i,j}$ to denote that the protection for a vulnerability of type v is installed within $b_{i,j}$ by setting $vi, j = 1$; otherwise, $vi, j = 0$. The proposed approach is applicable to a broad class of

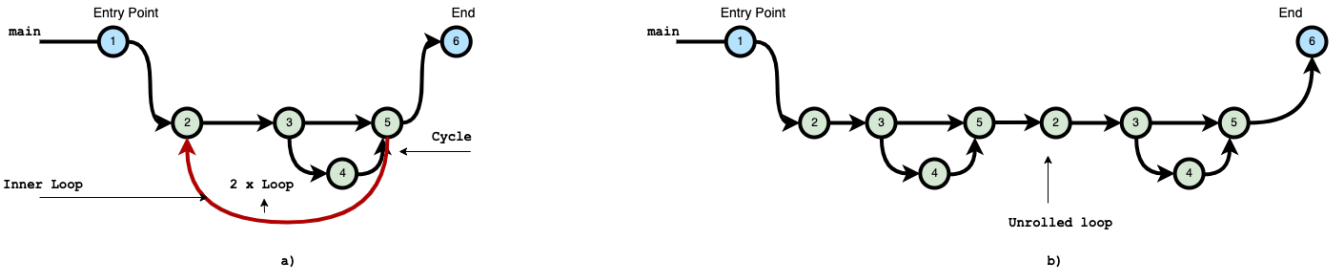

FIGURE 2. Example of CFG for a program before (inset (a)) and after (inset (b)) unrolling loops. Node labels correspond to the WCETs of the basic blocks.

vulnerability types provided that protections to be installed in BBs are available. The class of vulnerabilities and protections considered in this work are further detailed in Section III.

To quantify the security level of the application Γ with reference to the above security and threat model, the negative impact of a certain vulnerability type is quantified by means of a *security susceptibility score* (SSS), defined as a real value between 0 and 1, where larger values of the SSS correspond to a larger impact on the security of the application. For each vulnerability type $v \in \mathcal{V}$, the corresponding SSS is denoted by s^v, with $s^v \in [0, 1]$. A practical way of defining such SSSs for some relevant types of vulnerabilities is presented in Section III. In essence, the higher the value of s^v, the higher the benefit, in terms of security, of having the corresponding protection installed in a BB (i.e., $vi, j = 1$). For simplicity, this paper assumes a one-to-one correspondence between each vulnerability and each protection, meaning that a protection mechanism only protects from a single vulnerability, and a vulnerability can be protected only by a single protection mechanism.

D. IMPACT OF PROTECTIONS ON TIMING

Installing a protection mechanism in a BB comes at a cost in terms of real-time performance, since additional runtime overhead is introduced in the protected version of the BB. The overheads resulting from installing protection mechanisms in some of the BBs of a task produce an increase in the expected execution time of the task. The overhead contribution of the protection for a vulnerability of type v is assumed to be bounded by σ_v.

By leveraging the availability of the WCETs $C_{i,j}$ of each BB $b_{i,j} \in B_i$ of a task τ_i, it is possible to define the WCET $\overline{C}_i$ of τ_i when a number of protection mechanisms are installed in some of its BBs, as follows:

$$\overline{C}_i = \max_{\lambda \in \text{paths}(\tau_i)} \left\{ \sum_{b_{i,j} \in \lambda} \left(C_{i,j} + \sum_{v \in \mathcal{V}} vi, j \cdot \sigma_v \right) \right\}. \quad (1)$$

The maximum in Equation (1) is computed across all paths in paths(τ_i) to cover all possible execution flows of the program, thus accounting for the presence of control flow statements at the boundary of each BB.

E. QUANTIFYING THE SECURITY LEVEL

The overall security level S_Γ of the application Γ is quantified with an index given by a real value between 0 and 1 by computing the normalized weighted sum of the SSS of the protected BBs of each task, across all vulnerability types and all BBs, as follows:

$$S_\Gamma = \frac{\sum_{\tau_i \in \Gamma} \sum_{b_{i,j} \in B_i} \sum_{v \in \mathcal{V}} x^v_{i,j} \cdot s^v}{\sum_{\tau_i \in \Gamma} \sum_{b_{i,j} \in B_i} \sum_{v \in \mathcal{V}} s^v}. \quad (2)$$

The rationale behind Equation (2) is that the less BBs are protected, the more likely a successful attack on the program is. In general, an attacker may be capable of stimulating the execution of any path by providing appropriate inputs; hence, all the BBs are considered to contribute to the overall SSS of a task equally. Again, for the sake of simplicity, we consider that all BBs have the same probability of being vulnerable to a certain kind of attack. The model can be easily refined by introduce a per-BB weight in Equation (2) to differentiate the relevance of each BB to the overall security level according to properties of the corresponding code or even the likelihood of executing it (when available). For instance, for buffer overflow vulnerabilities the weights of the BBs involved in the access to a certain buffer could depend on the buffer size[1].

Under this definition, the highest security level ($S_\Gamma = 1$) for the application Γ is obtained when all protection mechanisms are installed in all the BBs of all tasks in Γ for each vulnerability type in $\mathcal{V}$. Conversely, a security level $S_\Gamma = 0$ means that no protection mechanism is installed in any of the basic blocks of the tasks composing the application Γ.

Table 1 summarizes the symbols used in the system model.

F. PROBLEM STATEMENT

Introducing additional protections for a task in the application Γ comes at the cost of increasing the expected execution time of the corresponding task. The aim of the proposed methodology is to optimize the security of the application Γ while ensuring that the timing constraints of all tasks in Γ are guaranteed to be met. In practice, this means that the proposed approach aims at maximizing the achieved security score S_Γ of the application, as defined in Equation (2), while ensuring that the WCRT R_i of all the tasks in Γ does not

[1]This is in line with compiler-assisted protections for stack overflows that avoid protecting functions that access small buffers.

TABLE 1. Symbols used in the system model.

Symbol	Description
m	Number of processors in the multiprocessor platform.
P_k	Processor k in the multiprocessor platform.
$P(\tau_i)$	Processor assigned to a task τ_i.
Γ	Application (or task set).
n	Number of tasks in the application Γ.
τ_i	Task i in the application Γ.
$\text{hp}_k(\tau_i)$	Set of tasks on P_k with higher priority than τ_i.
T_i	Minimum inter-arrival time of a task τ_i.
D_i	Relative deadline of a task τ_i.
C_i	BWCET of a task τ_i.
$C_{i,j}$	BWCET of a basic block $b_{i,j}$ of a task τ_i.
R_i	WCRT of a task τ_i.
G_i	CFG of a task τ_i.
E_i	Edges in the CFG G_i of a task τ_i.
B_i	Set of basic blocks of the program executed by a task τ_i.
$n_{i,B}$	Number of basic blocks of a task τ_i.
$b_{i,j}$	Basic block j of a task τ_i.
$\text{paths}(\tau_i)$	Paths in the CFG G_i of a task τ_i.
λ	Path in the set of paths $\text{paths}(\tau_i)$ in the CFG of a task τ_i.
$\mathcal{V}$	Set of vulnerability types.
v	Vulnerability type in the set of vulnerability types $\mathcal{V}$.
$x_{i,j}^v$	1 if $b_{i,j}$ is protected against vulnerability v; 0 otherwise.
s^v	SSS for a vulnerability type v.
σ_v	Protection overhead for a vulnerability type v.
$\overline{C}_i$	WCET of a task τ_i with protection mechanisms installed.
S_Γ	Overall security level of an application Γ.

exceed the corresponding deadline D_i, meaning that the application is schedulable.

III. SECURITY VULNERABILITIES AND PROTECTIONS

The security and threat model provided in Section II requires a characterization of the SSS value s^v for each protection in $\mathcal{V}$. This value quantifies the negative impact of the corresponding vulnerability on the security of a task in the application Γ, and, thus, also quantifies the positive impact on the security of the application given by installing the corresponding protection in the task.

To define SSS values, it is possible to refer to existing categorization and evaluation systems for security vulnerabilities. The Common Vulnerabilities and Exposures (CVE) program, managed by the MITRE Corporation with funding from the U.S. Department of Homeland Security (DHS) Cybersecurity and Infrastructure Security Agency (CISA), was introduced in 1999 to provide a common reference for known security vulnerabilities [19]. In particular, the CVE program identifies, defines, and catalogs vulnerabilities of publicly released software. By itself, the CVE cataloging system does not provide severity scoring or prioritization features. Another standard, the Common Vulnerability Scoring System (CVSS), operated by the Forum of Incident Response and Security Teams (FIRST), a global forum of collaboration for security teams, can be used to score the severity and prioritization of software vulnerabilities identified by CVE entries. CVSS provides a way to generate a numerical score reflecting the severity of a vulnerability by capturing its principal characteristics and the associated attack patterns as a way to aid prioritization for organizations involved in managing or analyzing the security of critical software and computing systems. The MITRE Corporation also manages the Common Weakness Enumeration (CWE) program, again funded by the CISA, to provide a categorization of the large number of threats cataloged in the U.S. National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), currently counting more than 200,000 CVE entries. The CWE program also leverages the CVSS scores associated with each CVE entry to evaluate the vulnerabilities and provide a baseline for weakness identification, analysis, and mitigation. The CWE program also maintains the CWE Top 25 Most Dangerous Software Weaknesses list (CWE Top 25), which reports the 25 most common and dangerous weaknesses (i.e., types of vulnerabilities) affecting publicly available software, updated on an annual basis to reflect the current security scenario.

The vulnerability classes presented in the CWE Top 25 are easily found in software and can be used to craft cyber-attacks of high severity, which enable the attacker to mount secondary attacks to access confidential information, disable a service, or take over the system completely. Entries in the CWE Top 25 are selected and ranked according to an aggregate score obtained by considering the number of CVE entries and the corresponding average CVSS score for each weakness collected as part of the CWE program. In particular, the evaluation of each weakness is performed with reference to a subset of CVE entries in the NVD, with a focus on entries corresponding to vulnerabilities that are known to have been exploited, collected in the CISA Known Exploited Vulnerabilities (KEV) catalog. The 2022 CWE Top 25 was assembled with reference to a total of 37,899 CVE entries from the previous two calendar years. The CWE Top 25 can be used by software maintainers as a guideline to ensure and assess the security of a piece of software by eliminating or mitigating as many weaknesses as possible among those in the list, and, subsequently, as a basis to formulate independent claims on the security of the product with a higher level on confidence. The score assigned to each entry in the CWE Top 25 ranges between 0 and 100, and represents the level of danger of the respective weakness, determined by multiplying a normalized severity score by a normalized frequency score calculated by considering the corresponding CVE entries in the dataset.

A. MATCHING CWE TOP 25 WITH OUR MODEL

The list of vulnerabilities in the CWE Top 25, together with the respective scores, represents a relevant set of software weaknesses that can be considered when building the set of vulnerability types $\mathcal{V}$ for optimizing the security level of a real-time application with the proposed approach. The advantages of leveraging this list when building the set $\mathcal{V}$ include the availability of a normalized score that relates to the level of danger of each weakness, which can be directly correlated to the concept of SSS s^v for a vulnerability type $v \in \mathcal{V}$. Then, each type of vulnerability can be associated with a corresponding set of software mitigations, to be applied at

TABLE 2. Selection of vulnerability classes for embedded software from the 2022 CWE Top 25, with corresponding score (normalized in $[0, 1]$).

Rank	CWE ID	Weakness short description	Score (s^v)
1	CWE-787	Out-of-bounds write	0.6420
4	CWE-20	Improper input validation	0.2063
5	CWE-125	Out-of-bounds read	0.1767
6	CWE-78	OS command injection	0.1753
7	CWE-416	Use after free	0.1550
11	CWE-476	NULL pointer dereference	0.0715
12	CWE-502	Deserialization of untrusted data	0.0668
13	CWE-190	Integer overflow or wraparound	0.0653
19	CWE-119	Improper restriction of operations	0.0485
22	CWE-362	Race condition	0.0357
23	CWE-400	Uncontrolled resource consumption	0.0356
25	CWE-94	Code injection	0.0332

the level of the basic block. Nonetheless, it should be noted that the proposed approach is general enough to support other quantitative scoring systems for software security, and is not necessarily tied to scores derived with the CVSS.

Table 2 reports the most significant CWE entries in the 2022 CWE Top 25 [20] related to the domain of embedded systems, in terms of rank, weakness identification number (CWE ID), vulnerability description (shortened, from the CWE database[2]), and overall score associated with the SSS parameter s^v (normalized in $[0, 1]$ by dividing the corresponding score in the 2022 CWE Top 25 by 100).

For each vulnerability type $v \in \mathcal{V}$, it is possible to assign the value of the protection cost parameter σ_v by evaluating the overhead introduced by applying the corresponding countermeasures for the vulnerability type v in a BB of the task. As a result, a protection pattern, usable at the level of the BB, should be defined for each vulnerability type $v \in \mathcal{V}$ and evaluated in terms of its overhead.

B. THE CASE FOR OUT-OF-BOUND WRITES AND READS

According to the CWE Top 25 ranking, the most critical software weakness is that of "*out-of-bounds write*". This weakness enables an attacker to change the values of memory areas located outside the memory area intended to be accessed by the software when writing contents in a data structure. This can result in data corruption or system failure and can also enable arbitrary code execution by the attacker. This weakness is related to the well-known stack-based buffer overflow and heap-based buffer overflow conditions. The CWE Top 25 also lists the related "*out-of-bounds read*" weakness, which affects a piece of software when the program can read data outside an intended memory area. This can allow an attacker to access sensitive information stored in other memory locations or cause a crash by attempting to access protected memory areas.

For instance, a basic condition resulting in a vulnerability of the class of out-of-bounds write occurs when using memory-unsafe programming languages (e.g., C/C++) with programs that accept an external input from the user to select the index of a buffer for a write operation, without checking whether the index is within the intended range of indexes. This notoriously allows an attacker to overwrite sensitive information stored in memory such as return addresses or function pointers [21]–[23], hence hijacking the control flow of the attacked task.

An effective mitigation for both out-of-bound writes and reads consists in checking that each memory access to a certain memory buffer is actually within the boundaries of the buffer itself. These checks are generally generated by compilers (e.g., when using memory-safe languages) [21, 23]–[26] and tend to have a significant impact on timing performance when extensively applied.

Measuring the timing overhead for these kinds of boundary checks is essential for estimating the increase in execution time of a protected basic block. Such an overhead can be estimated by means of static program analysis tools or by conservatively inflating the overheads observed with profiling tools to provide an upper bound on the actual execution time, as will be described in Section IV.

IV. MODEL EXTRACTION

The proposed optimization methodology for an application Γ, introduced in Section V, requires the availability of the BWCET C_i of all tasks $\tau_i \in \Gamma$, and the BWCET $C_{i,j}$ for all basic blocks $b_{i,j} \in B_i$. Static WCET analysis tools can be used to obtain such execution times. Similarly, WCET analysis can be applied to obtain the overhead introduced by applying a protection pattern for a vulnerability type $v \in \mathcal{V}$, in order to obtain an execution cost σ_v for the protection v.

Static WCET analysis tools, as an alternative to measurement-based WCET estimation [27], derive the WCET of a program by means of mathematical models of the timing behavior of a program when it is executed on a specific hardware architecture.

Several commercial and open-source tool suites implement static WCET analysis techniques. For instance, Heptane is an open-source static WCET analysis tool that leverages the Implicit Path Enumeration Technique (IPET) for static WCET estimation [15]. IPET analysis combines program flow and basic block execution time estimations into a set of constraints related to the structure of the program, represented by its CFG; then, each basic block and control flow is assigned an execution time coefficient and a counter variable, representing how many times that specific piece of code may be executed in the program. Finally, an upper bound on the WCET of the program is obtained by maximizing the product of the execution time coefficient and the counter variable, subject to the constraints imposed by the CFG.

For the purpose of evaluating the proposed approach on standard benchmarking suites for embedded software, we implemented a model extraction tool by building upon Heptane to obtain an upper bound for the execution time of each basic block in the program (corresponding to the BWCETs $C_{i,j}$ for the basic blocks $b_{i,j}$ in a task $\tau_i \in \Gamma$), in addition to the overall WCET of the program without protections (corresponding

[2]https://cwe.mitre.org/

to the BWCET C_i of a task $\tau_i \in \Gamma$). The custom tool also outputs the CFG of the program, together with the list of all possible paths in the CFG, obtained after unrolling the cycles in the CFG based on a safe estimation on the maximum number of times each cycle can be executed. In the system model, these paths correspond to the set of paths(τ_i) for a program $\tau_i \in \Gamma$, which is needed to estimate the WCET of the task τ_i when security protections are activated in the basic blocks of the corresponding program. Furthermore, in order to realize and evaluate a specialized analysis framework for the out-of-bounds write and out-of-bounds read vulnerability classes, presented in Section V-E, the tool can also extract the number of memory writes and memory reads performed within each basic block $b_{i,j}$ of all tasks $\tau_i \in \tau$.

V. OPTIMIZATION METHOD

This section presents an optimization approach that aims at finding a convenient trade-off between security and schedulability for an application Γ. The optimization problem is presented as a *mixed-integer linear programming* (MILP) formulation. The following are the primary goals of the proposed formulation:

- to determine, for each BBs of the tasks composing the application, whether it is convenient to install a protection mechanism for a vulnerability type v on that basic block, based on the benefit given in terms of security, which is evaluated by means of its SSS s^v, and on its cost, which is given by its overhead σ_v;
- to guarantee that each task will complete within its deadline despite the installation of protection mechanisms in some of its basic blocks.

Before introducing the actual formulation of the optimization problem, we report a practical schedulability test for fixed-priority scheduling that is particularly suitable for being encoded in MILP formulations.

A. SCHEDULABILITY ANALYSIS

The schedulability of a task set under preemptive fixed-priority scheduling can be assessed by means of the response-time analysis approach [28, 29]. With this approach, first the WCRT R_i of each task $\tau_i \in \Gamma$ is obtained; then, the task set is deemed schedulable if $R_i \leq D_i$ holds for all tasks, and not schedulable otherwise. Consider a task $\tau_i \in \Gamma$, statically allocated to processor $P(\tau_i) = P_k$, and assume that its WCET $\overline{C}_i$ was obtained by inflating the BWCET C_i to account for the overhead related to the protection mechanisms installed within the program of τ_i. The WCRT of the task τ_i is given by the least positive fixed point of the following recurrent equation [29]:

$$R_i = \overline{C}_i + \sum_{\tau_j \in \mathrm{hp}_k(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \overline{C}_j. \tag{3}$$

Note that determining the least positive fixed point of Equation (3), necessary to obtain the WCRT for a task τ_i in Γ, requires introducing integer variables to model the ceiling term [30], hence complicating the MILP formulation and introducing scalability issues in its solution.

By leveraging previous results by Park and Park [31], Pazzaglia et al. [30] derived a near-exact sufficient schedulability test that supports a much more efficient implementation within optimization problems. This test consists in checking a subset $\mathcal{Y}_i$ of the values in $[0, D_i]$ for each task $\tau_i \in \Gamma$, where the set of points $\mathcal{Y}_i$ is defined as follows [30]:

$$\mathcal{Y}_i = \{aT_j \mid \tau_j \in \{\mathrm{hp}_k(\tau_i) \cup \tau_i\} : a = \lfloor D_i/T_j \rfloor\} \cup \{D_i\}\}. \tag{4}$$

This reduces the schedulability conditions to just a very limited set of linear constraints. The accuracy drop with respect to the exact test in Equation (3) was experimentally found to be in the order of 1% on average [30].

The resulting test is then given by:

$$\exists y_{i,g} \in \mathcal{Y}_i \mid \overline{C}_i + \sum_{\tau_j \in \mathrm{hp}_k(\tau_i)} \left\lceil \frac{y_{i,g}}{T_j} \right\rceil \cdot \overline{C}_j \leq y_{i,g}. \tag{5}$$

If a point $y_{i,g}$ satisfies the above inequality, then it is a valid WCRT upper bound for τ_i.

Provided that the application Γ is schedulable according to the test in Equation (5) when no protection mechanism is installed in any of the basic blocks of its tasks (i.e., when the WCET $\overline{C}_i$ of each task τ_i is set to its BWCET C_i), the proposed MILP formulation will have at least one feasible solution, corresponding to the situation where no protection mechanism is applied. Instead, if the application without protections is deemed not schedulable, then the MILP formulation will have no feasible solution, since activating protections can only increase the response times of the tasks in the application.

B. VARIABLES

The proposed MILP formulation problem utilizes the following variables:

- For each $v \in \mathcal{V}$, for each $\tau_i \in \Gamma$, for each $b_{i,j} \in B_i$, $\mathrm{X}^v_{i,j} \in \{0, 1\}$ is a binary variable that is set to 1 if a protection mechanism for vulnerability v is installed in $b_{i,j}$, and is set to 0 otherwise.
- For each $\tau_i \in \Gamma$, $\mathrm{W}_i \in \mathbb{R}^{\geq 0}$ is a security-aware worst-case execution time bound for task τ_i, i.e., an upper-bound on the inflated WCET $\overline{C}_i$ of the task τ_i obtained by accounting for the overhead introduced by the protections installed in each BB of the task.
- For each $\tau_i \in \Gamma$, for each $y_{i,g} \in \mathcal{Y}_i$, $\mathrm{RTC}_{i,g} \in \mathbb{R}^{\geq 0}$ is a candidate worst-case response time upper bound for the task τ_i, needed for the implementation of the test in Equation (5) within the optimization problem.
- For each $\tau_i \in \Gamma$, for each $y_{i,g} \in \mathcal{Y}_i$, $\mathrm{E}_{i,g} \in \{0, 1\}$ is a binary variable set to 1 if $y_{i,g}$ is the candidate worst-case response time upper bound selected by the optimization problem among those encoded by variables $\mathrm{RTC}_{i,g}$.
- For each $\tau_i \in \Gamma$, $\mathrm{RT}_i \in \mathbb{R}^{\geq 0}$ is a worst-case response time upper bound of τ_i.

C. CONSTRAINTS

This section presents the set of constraints needed to compute worst-case response time bounds that are aware of the protection mechanisms installed in each BB. Constraint 1 bounds the inflated WCET $\overline{C}_i$ for each task τ_i, obtained by accounting for the overhead σ_v introduced due to the use of protection mechanisms.

Constraint 1 (Security-aware WCET upper bound)**.** $\forall \tau_i \in \Gamma,\ \forall \lambda \in \mathit{paths}(\tau_i)$,

$$W_i \geq \sum_{b_{i,j} \in \lambda} (C_{i,j} + \sum_{v \in \mathcal{V}} X^v_{i,j} \cdot \sigma_v). \tag{6}$$

It follows directly from Equation (1).

Constraint 2 establishes a WCRT upper bound for each task τ_i, when considering the WCET bounds determined within the variables W_i that account for the overhead σ_v introduced by the use of a protection mechanism. In the following, we leverage the so-called *big-M method*, where M is a large constant representing infinity, needed to implement the existential quantifier in Equation (5) and the selection of the WCRT upper bound among the candidate upper bounds in $\mathcal{Y}_i$.

Constraint 2 (Security-aware WCRT)**.** $\forall \tau_i \in \Gamma,\ \forall y_{i,g} \in \mathcal{Y}_i, P(\tau_i) = P_k$,

$$\mathit{RTC}_{i,g} \geq W_i + \sum_{\tau_j \in hp_k(\tau_i)} \left\lceil \frac{y_{i,g}}{T_j} \right\rceil \cdot W_j, \tag{7}$$

$$\mathit{RTC}_{i,g} \leq y_{i,g} + (1 - E_{i,g}) \cdot M, \tag{8}$$

$$\mathit{RT}_i \geq \mathit{RTC}_{i,g} - (1 - E_{i,g}) \cdot M. \tag{9}$$

$\forall \tau_i \in \Gamma$,

$$\sum_{y_{i,g} \in \mathcal{Y}_i} E_{i,g} = 1. \tag{10}$$

The first set of inequalities in Constraint (2) (Equation (7)) derives a WCRT upper bound candidate $\mathrm{RTC}_{i,g}$ for each point $y_{i,g} \in \mathcal{Y}_i$. The second set of inequalities (Equation (8), Equation (9), and Equation (10)) establishes exactly one WCRT upper bound candidate $\mathrm{RTC}_{i,g}$ as a valid WCRT upper bound for the task τ_i. In fact, the constraint in Equation (10) forces the MILP solver to select exactly one point $y_{i,g} \in \mathcal{Y}_i$ for which the corresponding variable $\mathrm{E}_{i,g}$ is going to be set to 1. If, for a given $y_{i,g} \in \mathcal{Y}_i$, $\mathrm{E}_{i,g} = 1$, then the inequality in Equation (9) enforces $\mathrm{RTC}_{i,g} \leq y_{i,g}$; otherwise, if $\mathrm{E}_{i,g} = 0$, the same inequality enforces $\mathrm{RTC}_{i,g} \leq \infty$, meaning that the constraint has no effect. Finally, the inequality in Equation (9) sets the WCRT upper bound of τ_i to be greater than or equal to the WCRT upper bound candidate corresponding to the point $y_{i,g} \in \mathcal{Y}_i$ for which $\mathrm{E}_{i,g} = 1$. This implementation of Equation (5) within the MILP formulation is analogous to the implementation presented in [32].

Constraint 3 enforces the schedulability of each task $\tau_i \in \Gamma$ by ensuring that the corresponding deadlines are always guaranteed to be respected.

Constraint 3 (Schedulability)**.** $\forall \tau_i \in \Gamma$,

$$\mathit{RT}_i \leq D_i. \tag{11}$$

If the task system with no protection installed is not schedulable according to the test in Equation (5), then it will not be possible to guarantee the satisfaction of both the constraints in Equation (9) and in Equation (11); therefore, the problem will have no feasible solution.

D. OBJECTIVE FUNCTION

The objective function maximizes the overall security score S_Γ of the task set, defined as in Equation (2):

$$\textbf{maximize} \quad \frac{\sum_{\tau_i \in \Gamma} \sum_{b_{i,j} \in B_i} \sum_{v \in \mathcal{V}} \mathrm{X}^v_{i,j} \cdot s^v}{\sum_{\tau_i \in \Gamma} \sum_{b_{i,j} \in B_i} \sum_{v \in \mathcal{V}} s^v}. \tag{12}$$

E. REFINED MODEL AND ANALYSIS

Given their high relevance in the CWE Top 25, in the following we present a specialized version of the model to protect against vulnerabilities of the type out-of-bounds write (CWE-787) and out-of-bounds read (CWE-125).

The model is modified to introduce an additional parameter $n^v_{i,j}$ for each task τ_i, each basic block $b_{i,j} \in B_i$, and each vulnerability type $v \in \mathcal{V}$, which represents the number of times the protection mechanism for the vulnerability type v is activated within the basic block $b_{i,j}$. For the specific case of out-of-bounds write and out-of-bounds read weaknesses, this parameter may be used to represent the number of memory writes, and the number of memory reads within the basic block $b_{i,j}$, respectively. Then, in this case, the meaning of the execution cost parameter σ_v of each protection type v is modified to represent the unit cost in terms of execution time relative to a single protection instance within the basic block. For the case of out-of-bounds write and out-of-bounds read vulnerabilities, this represents the overhead in terms of execution time introduced by protecting a single instance of buffer write or buffer read operations, respectively, within a generic basic block of a task, for instance, by performing additional boundary checks before writing to a buffer or reading from a buffer.

In this case, Constraint 1 is modified as follows to account for the updated model:

Constraint 4 (Security-aware WCET - Updated)**.** $\forall \tau_i \in \Gamma,\ \forall \lambda \in \mathit{paths}(\tau_i)$,

$$W_i \geq \sum_{b_{i,j} \in \lambda} (C_{i,j} + \sum_{v \in \mathcal{V}} X^v_{i,j} \cdot n^v_{i,j} \cdot \sigma_v). \tag{13}$$

F. COMPLEXITY ANALYSIS

Table 3 reports the complexity of the MILP formulation in terms of the number of constraints per type of constraint and the number of variables involved in each constraint type. In Table 3, M_P represents an upper bound on the maximum number of paths in $\mathrm{paths}(\tau_i)$ across all tasks $\tau_i \in \Gamma$. In addition, note that, for each task $\tau_i \in \Gamma$, the number of points in $\mathcal{Y}_i$ is $\mathcal{O}(n)$ [30]. The total complexity of the

TABLE 3. Complexity of the MILP formulation in terms of the number of constraints and number of variables per constraint.

Constraint	Eq.	# of constraints	# of variables per constraint
1	(6)	$\mathcal{O}(n \times M_P)$	$\mathcal{O}(\|\lambda\| \times \|\mathcal{V}\|)$
2	(7)	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
2	(8)	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
2	(9)	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
2	(10)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
3	(11)	$\mathcal{O}(n)$	$\mathcal{O}(1)$

MILP formulation in terms of the number of constraints is $\mathcal{O}(n^2 + n \times M_P)$.

VI. EXPERIMENTAL RESULTS

This section presents the results of an experimental evaluation of the proposed approach, carried out on a representative set of tasks included in a state-of-the-art benchmarking suite for embedded software.

A. BASELINE APPROACHES

We compared the proposed optimization approach with two greedy heuristic algorithms that determine suboptimal solutions to the investigated problem and that serve as a baseline for the evaluation.

Priority-based heuristic. The priority-based heuristic (PB-H) iterates over all tasks in the task set Γ in decreasing priority order. For each vulnerability type $v \in \mathcal{V}$, and for each basic block $b_{i,j} \in B_i$ of the current task τ_i, the protection for vulnerability v is activated for $b_{i,j}$ (i.e., $x^v_{i,j}$ is set to 1) if activating that protection (in addition to the protections that were already enabled by the heuristic) does not make the system unschedulable. The heuristic starts activating protections to a new task once all the protections of higher-priority tasks are enabled. The schedulability is determined according to the test in Equation (5), while the WCET of protected tasks is computed in accordance with the refined model presented in Section V-E.

Round-robin heuristic. Similar to PB-H, the round-robin heuristic approach (RR-H) iterates over all tasks in decreasing priority order and then iterates over vulnerability types and basic blocks. However, unlike PB-H, it only activates one protection for each task (if allowed by the schedulability test) before moving to the following one, in a round-robin fashion.

B. EVALUATED TASK SET

A selection of test programs from the Mälardalen WCET benchmarks [18] was adopted for the purpose of evaluating the proposed approach. These benchmarks consist of standard and specialized test programs for the evaluation of WCET estimation methodologies for a wide range of applications, and include a diverse range of program structures designed to aid testing of multiple program behaviors. Heptane includes a test platform based on the Mälardalen WCET benchmarks.

An application Γ composed of six tasks $\tau_1, \ldots, \tau_6$ was constructed for the evaluation, using varying system parameters across the experiments. The following test programs from the Mälardalen WCET benchmarks were considered as the program to be executed by each job of the corresponding task in Γ.

1) **lcdnum**: Read ten values, output half to LCD. Loop with the iteration-dependent flow.
2) **minver**: Inversion of floating point matrix. Floating value calculations in 3x3 matrix. Nested loops (3 levels)
3) **ns**: Search in a multi-dimensional array. Return from the middle of a loop nest, deep loop nesting (4 levels).
4) **bs**: Binary search for the array of 15 integer elements. Completely structured.
5) **insertsort**: Insertion sort on a reversed array of size 10. Input-data dependent nested loop with worst case of $(n^2)/2$ iterations (triangular loop).
6) **fibcall**: Iterative Fibonacci, used to calculate the Fibonacci series. Parameter-dependent function, single-nested loop.

Overall, these programs allow testing multiple program structures and constructs, including array and matrix computation, nested loops, input-dependent loops, inner loops depending on outer loops, floating-point computation, and bit manipulation.

The above test programs were analyzed with the static program analysis approach described in Section IV to obtain the WCET C_i, the set of paths paths(τ_i), and the WCET $C_{i,j}$ of each basic block $b_{i,j} \in B_i$ for each task $\tau_i \in \Gamma$. The execution time estimations for the experiments were performed with the ARM instruction set. In particular, the reference processor was the ARM7TDMI Reduced Instruction Set Computer (RISC) CPU for embedded systems, which implements the 32-bit ARMv4T architecture.

C. GENERATION OF SYSTEM PARAMETERS

The system parameters for the experiments were generated as follows. Let $U = \sum_{\tau_i \in \Gamma} C_i/T_i$ be the system utilization, and $U_i = C_i/T_i$ the utilization factor of each task τ_i. To avoid biasing effects, for a given system utilization U, the utilization factor U_i of each task $\tau_i \in \Gamma$ was generated by the UUniFast algorithm [33], in such a way that $\sum_{\tau_i \in \Gamma} U_i = U$. Then, the minimum inter-arrival time for each task was assigned as $T_i = C_i/U_i$, while the deadline D_i was selected from a discrete uniform distribution in the range $[C_i + (T_i - C_i) \cdot \alpha, T_i]$, where α is a generation parameter such that $\alpha \in [0, 1]$ so that $C_i \leq D_i \leq T_i$ (constrained deadlines). Note that if $\alpha = 1$, then the relative deadline is assigned as $D_i = T_i$ (implicit deadlines).

An additional parameter $rand$ was introduced to represent the fact that not all basic blocks in a program may be subject to a certain vulnerability or applicable for a corresponding protection mechanism. In particular, a certain percentage of basic blocks in B_i for each task in Γ, selected randomly up to the percentage given by the parameter $rand$, were considered as not being applicable for a protection mechanism

and discarded from the optimization procedure, meaning that the protections corresponding to those basic blocks were statically disabled. This parameter also serves for introducing a diversity among the task sets evaluated in different iterations of the experiments, in addition to the minimum inter-arrival time and deadline parameters. Note that, when computing the security level of the application Γ, defined in Equation (2), the contribution of the basic blocks discarded from the optimization by means of the $rand$ parameter is not accounted for when considering the resulting security score of the application.

The experiments considered a platform composed of a single processor P_1, and all tasks in Γ were assigned to processor P_1. Task priorities were set based on the Deadline Monotonic algorithm, which assigns higher priority levels to tasks with smaller relative deadline D_i.

D. VULNERABILITY PARAMETERS

Two representative vulnerability types were considered in the experiments, one related to the out-of-bounds write weakness, and one related to the out-of-bounds read weakness in the 2022 CWE Top 25, presented in Section III. For each vulnerability type v, the corresponding SSS s_v was assigned based on the score attributed to the corresponding weakness in the 2022 CWE Top 25, as reported in Table 2; that is, the SSS of the vulnerability type corresponding to the out-of-bounds write weakness was set to 0.6420, while the SSS of the vulnerability type corresponding to the out-of-bounds read weakness was set to 0.1767.

For the purpose of carrying out the experimental evaluation, a protection mechanism implemented in the C language was considered for both vulnerability types. As discussed in Section III, this protection mechanism consists in checking that the index to be accessed with a write or read access to a buffer translates to a valid location inside the memory buffer. If the condition is not satisfied, a security violation is triggered, and the program is terminated, otherwise the buffer access operation is performed normally. The cost of applying this kind of protection mechanism for one buffer access of the corresponding type was evaluated with Heptane (under the same settings used for analyzing the tasks), resulting in worst-case overheads of 20 time units (processor cycles). Therefore, the protection costs σ_v were set to 20 for each $v \in \mathcal{V}$.

The corresponding protection overheads in terms of execution times utilized for the experiments for the out-of-bounds write and out-of-bounds read vulnerability types are reported in Table 4.

TABLE 4. Vulnerability types considered in the experiments.

Rank	CWE ID	Vulnerability Name	Protection	σ_v
1	CWE-787	Out-of-bounds write	Boundary check	20
5	CWE-125	Out-of-bounds read	Boundary check	20

The number of times a protection v can be activated for a basic block $b_{i,j}$ of a task τ_i, given by the parameter $n^v_{i,j}$, corresponds in this case to the number of memory writes and memory reads performed within the basic block $b_{i,j}$, with reference, respectively, to the out-of-bounds write and out-of-bounds read vulnerability types. In the experiments, the value of $n^v_{i,j}$ for each basic block $b_{i,j}$ of all tasks $\tau_i \in \Gamma$ was extracted using the custom analysis tool based on Heptane, presented in Section IV.

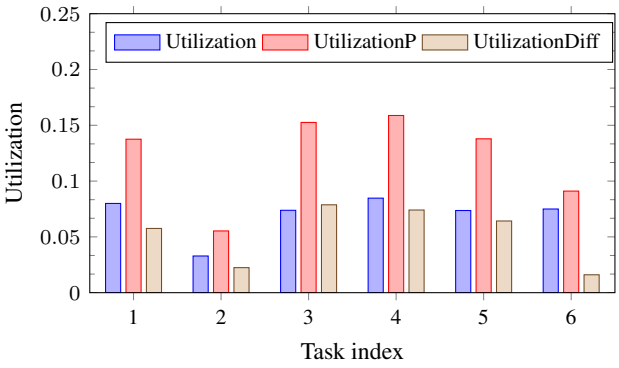

FIGURE 3. Evaluation of utilization levels of each task in an application Γ.

E. CHARACTERIZATION OF TASK SET GENERATION

The first set of experiments evaluated the relevance of applying the proposed optimization approach to an application Γ generated according to the proposed generation methodology, by comparing the difference in terms of utilization of the generated task sets introduced when additional security protections are installed. Figure 3 reports the results of an experiment that analyzes the utilization value obtained for each task in a single application Γ when no protection is active (Utilization) and when all protections are active for all basic blocks in the corresponding task (UtilizationP), with respect to a representative system utilization value $U = 0.42$ for the case of implicit deadlines ($\alpha = 1$), with $rand = 0.2$. The difference between the two utilizations (UtilizationP minus Utilization) is also reported (UtilizationDiff). These results show that the decisions to be performed by the optimization method have a large impact on the final utilization of the system, essential in order to optimize the security level of the application while maintaining its schedulability.

In the following, let ΔU represent the utilization gap for an application Γ, defined as

$$\Delta U = \sum_{\tau_i \in \Gamma} U^p_i - U_i, \tag{14}$$

where U^p_i is the utilization factor for the task τ_i obtained when all protection mechanisms available for the basic blocks of τ_i are installed. The utilization gap metric is useful in order to evaluate the potential system utilization difference introduced by applying all the protections available for the application Γ. In particular, the larger the value of the utilization gap, the higher the relevance of the security protection overheads with respect to the overall system utilization.

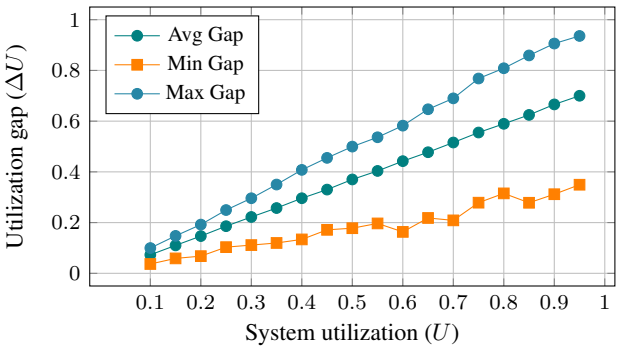

FIGURE 4. Utilization gap ΔU obtained when varying the system utilization U.

The cases where the value of the utilization gap is higher correspond to situations in which applying the proposed optimization approach can be considered more relevant and advantageous. In fact, performing a fine-grained decision on whether to apply each protection mechanism to each basic block in the application has a higher impact on the resulting schedulability. Figure 4 reports the average, minimum, and maximum value of the utilization gap ΔU obtained over 1000 generated task sets when varying the system utilization U from 0 to 1 in increments of 0.05. This experiment considers the case of implicit deadlines ($\alpha = 1$), with $rand = 0.2$.

F. SCHEDULABILITY RATIO AND SECURITY LEVEL

In the following set of experiments, the value of the system utilization parameter U was varied from 0 to 1 with step 0.05. For each value of the system utilization U, 1000 task sets were generated. For each task set Γ, the schedulability of Γ was first verified using the schedulability test by Pazzaglia et al. [30] and, if Γ was deemed schedulable by the test, the optimization procedure and the PB-H and RR-H heuristic approaches were applied to Γ. These experiments evaluated the schedulability ratio and the average security level of each generated application. The schedulability ratio corresponds to the number of task sets deemed schedulable when applying the schedulability test by Pazzaglia et al. [30], over the number of task sets generated for a specific value of the system utilization U. The average security level for a specific value of the system utilization U was obtained by averaging the optimal value of the proposed optimization problem or the security level S_Γ obtained with the PB-H and RR-H heuristics (based on Equation (12) or Equation (2), respectively) among the task sets that were deemed schedulable. Note that the optimal value obtained by applying the optimization approach to a schedulable application Γ corresponds to the security level S_Γ of the application when the protections selected by the optimization approach are installed in the application.

Figure 5 reports the result of these experiments for various representative system configurations. The value of the parameters α and $rand$ are reported above each graph. In particular, Figures 5(a)-(b) report the results for the case of implicit deadlines ($\alpha = 1$), while Figures 5(c)-(d) report the results for the case of constrained deadlines ($\alpha = 0.4$). A common trend among the results is that the schedulability ratio decreases for system utilization values higher than approximately 0.7, which is expected for task systems executing under fixed-priority scheduling policies [34]. In addition, the value of the average security level decreases with higher system utilization, meaning that the evaluated approaches are forced to keep some of the protection mechanisms disabled for some of the basic blocks of the tasks composing the application Γ, in order to keep the application schedulable. This is due to the fact that higher utilization values cause higher interference on lower priority tasks, resulting in an unschedulable task set when all protection mechanisms are installed in all basic blocks of the application. On the other hand, a security level of 1 is achieved for lower values of the system utilization, meaning that the task set can tolerate an increase in execution time up to a certain threshold before the optimizer and the heuristics are forced to keep some of the protections disabled for at least some basic block. As expected, both heuristics achieve lower performance values than the MILP approach, given that they typically perform suboptimal choices. Among the two heuristics, RR-H performs the closest to the optimization approach, while PB-H exhibits a steeper performance drop. The case of constrained deadlines (Figures 5(c)-(d)) shows a faster schedulability loss due to the tighter deadlines imposed for the tasks in the system. The fact that the deadlines are tighter in this case also has an impact on the achieved average security level. In fact, in this case, the optimizer and the heuristics are forced to keep even more protections disabled to meet the tighter timing constraints. When the value of $rand$ is increased from $rand = 0.2$ to $rand = 0.5$, the results show a marginal increase in the security level achieved with the MILP approach. This is due to the fact that achieving a certain security level when a smaller number of basic blocks in a task are vulnerable (i.e., when the $rand$ parameter has a higher value) requires installing protection mechanisms in less basic blocks. Additionally, in this case, the PB-H heuristic shows a larger performance gain than the RR-H heuristic.

G. OTHER RESULTS

This section presents additional experimental results that show how the schedulability ratio and the security level of the application change when the protection costs are increased. In addition, results concerning the runtimes of the evaluated algorithms are reported.

The experimental results reported in Figure 6 show how the average security level obtained by applying the evaluated approaches varies with higher values of the protection cost σ_v with respect to a fixed system utilization value $U = 0.42$ for the case of implicit deadlines ($\alpha = 1$), with $rand = 0.2$. In the experiment, the value of σ_v is varied from 0 to 1000, in increments of 50 time units, and a total of 1000 task sets were generated for every such value of σ_v. Again, we applied

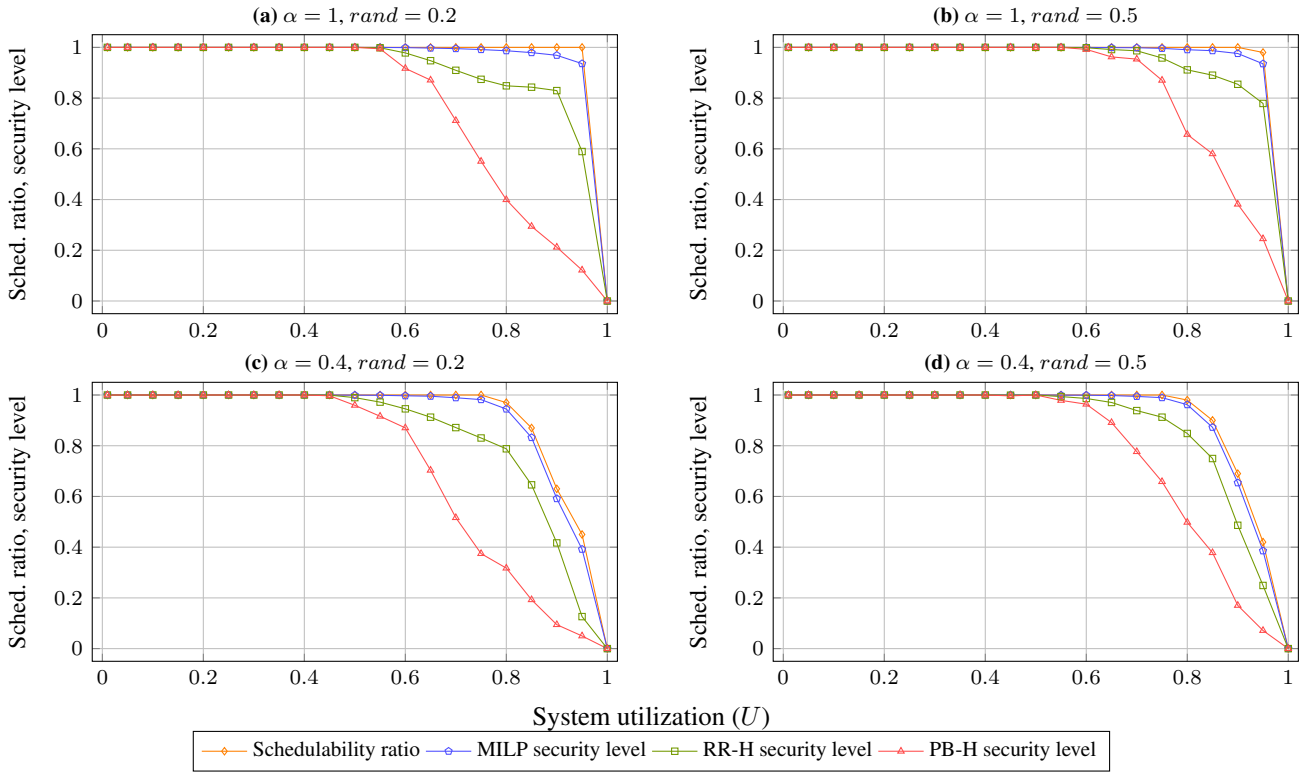

FIGURE 5. Schedulability ratio and average security level for applications generated with different system configurations.

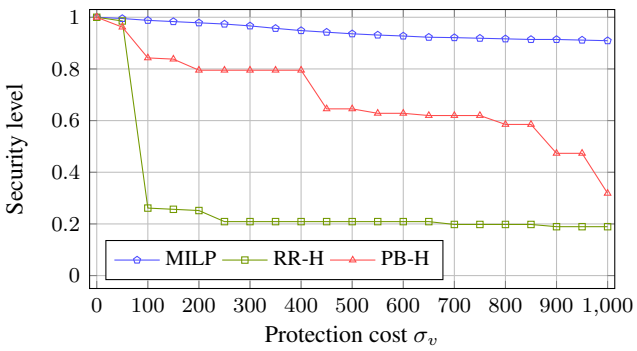

FIGURE 6. Security level obtained when varying the protection cost σ_v.

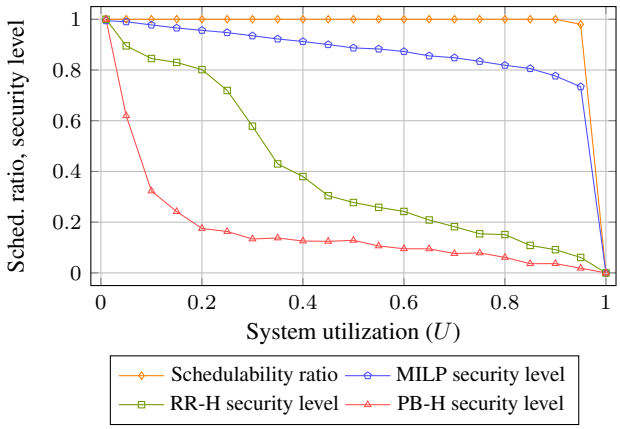

FIGURE 7. Schedulability ratio and average security level for applications generated with different system utilization.

the test by Pazzaglia et al. [30] to assess schedulability, and all task sets resulted in being schedulable for this experiment. As expected, the results show that the security level attained after the proposed optimization approach is applied decreases with an increase in the execution cost of applying a protection mechanism to a basic block. Moreover, this experiment highlights a drastic performance loss for both the PB-H and the RR-H heuristics with larger values of σ_v, with respect to the results obtained with the MILP approach. This is arguably due to the fact that the greedy decisions performed by the heuristics have a larger impact on future decisions when the cost of each protection is higher.

Figure 7 illustrates the average security level and the schedulability ratio as a function of the system utilization, obtained with $\alpha = 1$ and $rand = 0.2$ when the cost σ_v of the two evaluated protections $v \in \mathcal{V}$ is set to 1000 time units. In this case, the PB-H and RR-H heuristics show a steep performance drop starting with small values of the system utilization U, while the MILP approach yields a more robust performance in terms of the achieved security level.

Figure 8 reports statistics on the runtime of the proposed optimization and heuristic approaches, measured when performing the experiment in Figure 5(a) on a computer

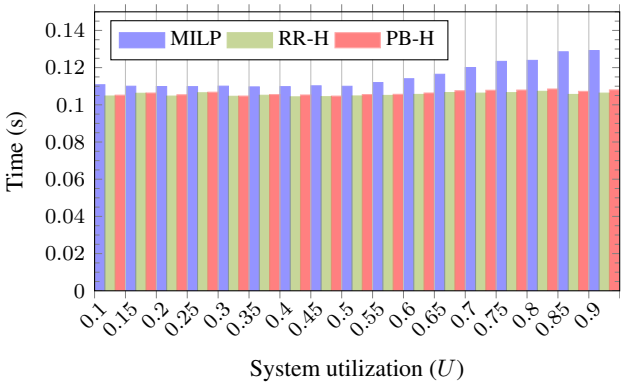

FIGURE 8. Average runtimes collected in the experiment in Figure 5(a).

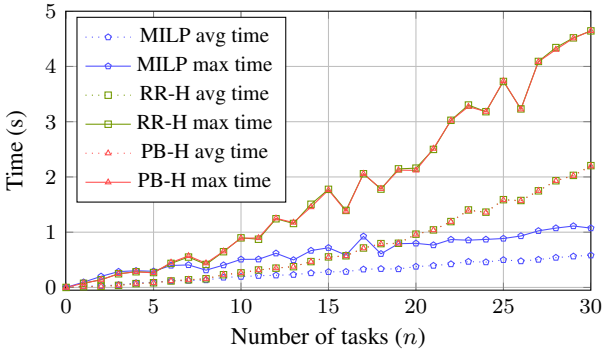

FIGURE 9. Average and maximum runtimes obtained when varying the number of tasks.

equipped with an Intel Core i9-9900 processor with 8 multithreaded cores running at a base operating frequency of 3.10 GHz, and 32 GB of DRAM. The average runtimes of the proposed optimization approach are reported for each value of the system utilization U. Overall, the runtimes of the approaches are in the order of tenths of seconds.

Finally, Figure 9 reports the average and maximum runtimes for the evaluated approaches obtained when varying the number of tasks n in Γ from 1 to 30, with $U = 0.42$, $\alpha = 1.0$, and $rand = 0.2$. In this experiment, the n tasks in the application Γ were randomly selected among the six benchmarks considered in the evaluation (with an equal chance); then, the task parameters were generated as in Section VI-C. These results show that the MILP approach scales better than the PB-H and RR-H heuristics with respect to the number of tasks in the application, while the two heuristics perform similarly, given that they execute the same amount of schedulability assessments. Overall, these results show that all the evaluated approaches can efficiently be applied off-line during the system design phase. However, in such off-line design scenarios, the MILP approach should be preferred due to the higher security level it can achieve.

VII. RELATED WORK

This section reviews related works in the literature concerning both real-time systems and security from a broad perspective.

Trade-offs between security and schedulability. A number of existing works aim at integrating security features into real-time systems by means of heuristic or optimal techniques that improve the security of the system while maintaining its schedulability. On this front, Xie and Qin [35] proposed a security overhead model for security-critical tasks in real-time applications, which provides a quantitative measure for the evaluation of overheads related to security services, coupled with a security-aware scheduling strategy incorporating the Earliest Deadline First (EDF) policy. Lin et al. [36] extended this methodology by adopting a group-based security approach where security services are partitioned into groups, combined with a scheduling algorithm based on the EDF policy. Then, an associated security-aware schedulability analysis is provided, while optimization techniques are explored to select the best combination of security services while guaranteeing the schedulability of the system. Roy et al. [37] considered the problem of maximizing the quality level of software systems with multiple implementations producing solutions with different degrees of accuracy while still finding a feasible schedule. Similar to this paper, it finds a trade-off between schedulability and another requirement, but it does not target security. Lesi et al. [38] proposed a method to improve the quality of control in the presence of security attacks on sensor measurements that undermine data integrity, focusing on real-time control tasks and EDF scheduling. Differently, this work focuses on fixed-priority scheduling, and it is not limited to control tasks and integrity-based attacks; instead, it tackles the problem from a more general perspective. Mohan et al. [39] proposed a methodology to integrate security requirements into real-time fixed-priority scheduling algorithms, with reference to security issues related to information leakage via shared resources. The methodology implements additional constraints on the scheduling algorithm as a way to mitigate potential security issues, and is coupled with an analysis of the resulting scheduling overheads. Hasan et al. [40] focused on the integration of security mechanisms into real-time systems, by combining opportunistic execution of security tasks with hierarchical scheduling. The proposed technique is based on an optimization problem, which guarantees the schedulability of the tasks while maximizing the security of the system with reference to a specialized security metric. Hasan et al. [41] proposed an optimization approach for task-to-core allocation of security tasks in multiprocessor systems that leverages opportunistic execution to maximize the security of the system while preserving schedulability. Völp et al. [42] introduced variant versions of standard real-time resource locking protocols that preserve the confidentiality guarantees of the underlying schedulers, thus preventing information leakage while preserving the related timing guarantees.

While these papers share the purpose of balancing the trade-off between security and schedulability with this work, none of them focus on enabling security protections at the level of the basic blocks, rather than introducing additional security services into the task set.

Security at the basic-block, OS, and hypervisor level. Other related research was carried out to improve the security of real-time systems at the level of the basic block or by modifying the operating system and hypervisor-level scheduler and services. Fellmuth et al. [43] presented a safe approach for artificial software diversity in safety-critical real-time systems aimed at preventing code-reuse attacks while preserving the timing properties of real-time tasks. The approach leverages software diversity at the level of the basic blocks and employs static WCET analysis techniques to control the impact on the resulting WCETs. Specialized algorithms are provided to determine the diversification strategy. Singh et al. [44] proposed a new scheduling algorithm called RT-SANE to address security and real-time requirements on fog networks. However, different from this work, it focuses on online strategies to handle the submission of non-recurring jobs in a distributed system. Chai et al. [45] presented a short review of different security-aware techniques used in real-time embedded systems. Designers of safety-critical real-time systems need to be aware of attack vectors as they can leak important temporal information, such as future arrival times of real-time tasks. Chen et al. [46] presented an algorithm that demonstrates how to exploit scheduler side channels in fixed-priority real-time systems.

Despite threads having access to exact times, a change to fixed-priority schedulers allows them to bypass timing channels. A fixed-priority budget enforcer is changed to handle active threads that block or stop early and might potentially leak information as if they were ready. While this lengthens the system's idle time, information can no longer be accessed and some threads results isolated from the behavior of others. Völp et al. [47] demonstrated how conventional techniques may be employed to achieve real-time guarantees for a modified scheduler. The technique beats time-partitioning schedulers in terms of obtained real-time guarantees while giving equivalent isolation properties.

However, isolation is not a suitable method if applied alone, as demonstrated by Mergendahl et al. [48]. The addition of priority and budget awareness to IPC processing can cause severe interference due to the kernel's prioritization mechanism, as demonstrated in this study. As a result, temporal isolation is required to reduce the impact of malicious software. Multiple client threads using IPC to seek service from a shared server will affect each other's response times. A Thundering Herd of malicious threads can dramatically delay the activation of mission-critical processes in both scenarios. Priority-sorted endpoint queues and replenishment strategies are common mitigations for these attacks. Unfortunately, the authors discovered that these technologies could be used to launch new and powerful attacks. Borgioli et al. [49] proposed an I/O virtualization mechanism resilient to denial-of-service (DoS) attacks due to I/O-related memory traffic, leveraging the QoS-400 regulators by Arm [50].

Jero et al. [51] presented Patina, a prototypical real-time operating system API that supports ubiquitous functions in feature-rich OSes, but not in more reliable kernel-based systems. Two Patina implementations are shown, one on Composite and the other on seL4, which aim at boosting system security using the Principle of Least Privilege (PoLP). The evaluations show that the performance of the PoLP-based version of Patina is equivalent to or better than Linux, while also providing increased isolation.

Son et al. [52] presented a mathematical framework for investigating the influence of the Rate Monotonic real-time scheduling policy on covert timing channels. They demonstrated that in some system configurations, it is not feasible to close the covert channel totally and that High and Low can use the noisy channel for secret communication. They suggested a straightforward way to develop a security metric that compares covert channels in terms of the relative amount of probable information leakage by comparing the degree of deducibility of one covert channel to another.

Nevertheless, none of these works focus on optimization methods to balance security and schedulability.

Security at the hardware level. Other work focuses on enforcing security at the micro-architectural level, either by considering attacks that can occur due to the design of the hardware platform, or leveraging the hardware itself (e.g., through specialized devices implemented in FPGA) to protect from cyber-attacks.

Serra et al. [53] presented PAC-PL, a hardware-assisted solution to enable CFI in FPGA-equipped SoCs. PAC-PL consists mainly of a hardware accelerator deployed in programmable logic that allows cryptographically signing and authentication of pointers to enforce their integrity. The hardware accelerator is accompanied by a compiler plugin that recognizes the storage of pointers in memory (such as return addresses pushed into the stack) generating instructions to sign those pointers and authenticate them on usage.

Bechtel and Yun [54] presented an operating-system-level method to avoid denial of service (DoS) attacks on multi-core systems with shared caches. They extended a memory bandwidth management technique to minimize cache DoS attacks on the shared cache's writeback buffer. They found that the shared cache's writeback buffer is a viable DoS attack vector. Hence, their approach implements a distinct read-and-write memory bandwidth control technique to effectively resist write-intensive cache DoS attacks while reducing performance implications on read-heavy regular workloads.

Differently, the Aker [55] design and verification framework adopts a hardware module - called Access Control Wrapper - to manage the accesses to shared resources. The access control system is guaranteed to be safe and secure by a property-driven security verification leveraging MITRE CWE, which is also used in this work. Other works leveraging custom hardware modules to enhance security, e.g.,

to protect the system from attacks occurring at the bus level [56]–[58].

However, all these works target attacks that cannot be tackled at the level of the basic block, and hence they are orthogonal and complementary to this paper.

Overall, to the best of our knowledge, no prior work derived the optimal trade-off between security and schedulability, considering the susceptibility of tasks to security vulnerabilities and fixed-priority real-time scheduling.

VIII. CONCLUSIONS AND FUTURE WORK

This paper presented a methodology to optimize the security of real-time software while preserving the schedulability of the system in terms of timing analysis. The approach leverages an MILP formulation to maximize the security level of each task in the system by applying additional protection mechanisms at the level of the basic block of the corresponding program, while ensuring the system schedulability by applying an efficient response-time analysis approach within the optimization problem. The performance and efficiency of the proposed approach were assessed in an experimental evaluation that was carried out on a representative set of tasks from standard benchmarking suites, with reference to protection mechanisms that shield the software against critical weaknesses of embedded software, namely, out-of-bounds write and out-of-bounds read conditions.

The flexibility and modularity of the proposed optimization problem gives ample possibilities for extension in future work. Directions for future extensions include considering a refined model to account for complex protections distributed across basic blocks, accounting for end-to-end delays of task chains [59, 60], accounting for the EDF scheduling policy in the optimization, developing a complete tool chain for security optimization of real-time software, and specializing the optimization problem to other relevant vulnerability classes from the CWE rankings.

ACKNOWLEDGMENT

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

[1] S. Park, J. Choi, S. Hwang, and C.-G. Lee, "Ros2 extension of functionally and temporally correct real-time simulation of cyber systems for automotive systems," ser. ISEEIE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 185–189.

[2] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 41–53.

[3] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[4] C. Bédard, P.-Y. Lajoie, G. Beltrame, and M. Dagenais, "Message flow analysis with complex causal links for distributed ros 2 systems," *Robotics and Autonomous Systems*, p. 104361, 2023.

[5] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[6] A. Hamann, S. Saidi, D. Ginthoer, C. Wietfeld, and D. Ziegenbein, "Building end-to-end IoT applications with QoS guarantees," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[7] L. Belluardo, A. Stevanato, D. Casini, G. Cicero, A. Biondi, and G. Buttazzo, "A multi-domain software architecture for safe and secure autonomous driving," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021, pp. 73–82.

[8] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated cpu/gpu memory management, an implementation on autonomous driving platform," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 310–323.

[9] K. Ahmad, M. Maabreh, M. Ghaly, K. Khan, J. Qadir, and A. Al-Fuqaha, "Developing future human-centered smart cities: Critical analysis of smart city security, data management, and ethical challenges," *Computer Science Review*, vol. 43, p. 100452, 2022.

[10] P. M. Rao and B. Deebak, "Security and privacy issues in smart cities/industries: Technologies, applications, and challenges," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–37, 2022.

[11] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[12] G. Culot, F. Fattori, M. Podrecca, and M. Sartor, "Addressing industry 4.0 cybersecurity challenges," *IEEE Engineering Management Review*, vol. 47, no. 3, pp. 79–86, 2019.

[13] R. A. Martin and S. Barnum, "A status update: The common weaknesses enumeration," *Proc. of the Static Analysis Summit (NIST Special Publication 500-262)*, pp. 62–64, 2006.

[14] Z. Shao, C. Xue, Q. Zhuge, M. Qiu, B. Xiao, and E.-M. Sha, "Security protection and checking for embedded system integration against buffer overflow attacks via hardware/software," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 443–453, 2006.

[15] D. Hardy, B. Rouxel, and I. Puaut, "The heptane static worst-case execution time estimation tool," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[16] Y. Sun and M. Di Natale, "Pessimism in multicore global schedulability analysis," *Journal of Systems Architecture*, vol. 97, pp. 142–152, 2019.

[17] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 99–110.

[18] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The mälardalen wcet benchmarks: Past, present and future," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[19] D. E. Mann and S. M. Christey, "Towards a common enumeration of vulnerabilities," in *2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, Indiana*, 1999.

[20] A. Gueye, C. E. Galhardo, I. Bojanova, and P. Mell, "A decade of reoccurring software weaknesses," *IEEE Security & Privacy*, vol. 19, no. 6, pp. 74–82, 2021.

[21] C. Cowan, F. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer overflows: Attacks and defenses for the vulnerability of the decade," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2. IEEE, 2000, pp. 119–129.

[22] K.-S. Lhee and S. J. Chapin, "Buffer overflow and format string overflow vulnerabilities," *Software: practice and experience*, vol. 33, no. 5, pp. 423–460, 2003.

[23] D. Larochelle and D. Evans, "Statically detecting likely buffer overflow vulnerabilities," in *2001 USENIX Security Symposium, Washington, DC*. Version: http://www. usenix. org/events/sec01/larochelle. html, 2001.

[24] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, "Softbound: Highly compatible and complete spatial memory safety for c," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2009, pp. 245–258.

[25] D. Dhurjati, S. Kowshik, V. Adve, and C. Lattner, "Memory safety without runtime checks or garbage collection," in *Proceedings of the 2003 ACM*

SIGPLAN conference on language, compiler, and tool for embedded systems, 2003, pp. 69–80.
[26] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer, "Rustbelt: Securing the foundations of the rust programming language," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, pp. 1–34, 2017.
[27] R. Lavaee, J. Criswell, and C. Ding, "Codestitcher: inter-procedural basic block layout optimization," in *Proceedings of the 28th International Conference on Compiler Construction*, 2019, pp. 65–75.
[28] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.
[29] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.
[30] P. Pazzaglia, A. Biondi, and M. D. Natale, "Simple and general methods for fixed-priority schedulability in optimization problems," in *Proceedings of the International Conference on Design, Automation and Test in Europe (DATE 2019)*, March 2019.
[31] M. Park and H. Park, "An efficient test method for rate monotonic schedulability," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1309–1315, 2014.
[32] D. Casini, P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration," *Journal of Systems Architecture*, vol. 124, p. 102416, 2022.
[33] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.* IEEE, 2004, pp. 196–203.
[34] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.
[35] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," *IEEE transactions on computers*, vol. 55, no. 7, pp. 864–879, 2006.
[36] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, pp. 22–37, 2009.
[37] S. K. Roy, R. Devaraj, A. Sarkar, and D. Senapati, "Slaqa: Quality-level aware scheduling of task graphs on heterogeneous distributed systems," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, jul 2021.
[38] V. Lesi, I. Jovanov, and M. Pajic, "Security-aware scheduling of embedded control tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–21, 2017.
[39] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Integrating security constraints into fixed priority real-time schedulers," *Real-Time Systems*, vol. 52, no. 5, pp. 644–674, 2016.
[40] M. Hasan, S. Mohan, R. B. Bobba, and R. Pellizzoni, "Exploring opportunistic execution for integrating security into legacy hard real-time systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 123–134.
[41] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba, "A design-space exploration for allocating security tasks in multicore real-time systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 225–230.
[42] M. Völp, B. Engel, C.-J. Hamann, and H. Härtig, "On confidentiality-preserving real-time locking protocols," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013, pp. 153–162.
[43] J. Fellmuth, P. Herber, T. F. Pfeffer, and S. Glesner, "Securing real-time cyber-physical systems using wcet-aware artificial diversity," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2017, pp. 454–461.
[44] A. Singh, N. Auluck, O. Rana, A. Jones, and S. Nepal, "Scheduling real-time security aware tasks in fog networks," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1981–1994, 2019.
[45] H. Chai, G. Zhang, J. Zhou, J. Sun, L. Huang, and T. Wang, "A short review of security-aware techniques in real-time embedded systems," *Journal of Circuits, Systems and Computers*, vol. 28, no. 02, p. 1930002, 2019.
[46] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 90–102.
[47] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, 2008, pp. 44–55.
[48] S. Mergendahl, S. Jero, B. C. Ward, J. Furgala, G. Parmer, and R. Skowyra, "The thundering herd: Amplifying kernel interference to attack response times."
[49] N. Borgioli, M. Zini, D. Casini, G. Cicero, A. Biondi, and G. Buttazzo, "An i/o virtualization framework with i/o-related memory contention control for real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4469–4480, 2022.
[50] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling i/o-related memory contention in cots heterogeneous platforms," *Software: Practice and Experience*, vol. 52, no. 5, pp. 1095–1113, 2022.
[51] S. Jero, J. Furgala, R. Pan, P. K. Gadepalli, A. Clifford, B. Ye, R. Khazan, B. C. Ward, G. Parmer, and R. Skowyra, "Practical principle of least privilege for secure embedded systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 1–13.
[52] J. Son *et al.*, "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems," in *2006 IEEE Information Assurance Workshop*. IEEE, 2006, pp. 361–368.
[53] G. Serra, P. Fara, G. Cicero, F. Restuccia, and A. Biondi, "Pac-pl: Enabling control-flow integrity with pointer authentication in fpga soc platforms," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 241–253.
[54] M. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 357–367.
[55] F. Restuccia, A. Meza, and R. Kastner, "Aker: A design and verification framework for safe and secure soc access control," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.
[56] Z. Jiang, K. Yang, Y. Ma, N. Fisher, N. Audsley, and Z. Dong, "I/o-guard: Hardware/software co-design for i/o virtualization with guaranteed real-time performance," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1159–1164.
[57] F. Restuccia, A. Biondi, M. Marinoni, and G. Buttazzo, "Safely preventing unbounded delays during bus transactions in fpga-based soc," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 129–137.
[58] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
[59] D. Casini and A. Biondi, "Placement of chains of real-time tasks on heterogeneous platforms under edf scheduling," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 149–156.
[60] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Analyzing end-to-end delays in automotive systems at various levels of timing information," *SIGBED Rev.*, vol. 14, no. 4, p. 8–13, jan 2018.

SANDRO DI LEONARDI is a Ph.D. student in Emerging Digital Technologies - curriculum Embedded Systems, at the Real-Time System Laboratory (ReTiS), of the Scuola Superiore Sant'Anna of Pisa, under the supervision of Prof. Giorgio Buttazzo and Prof. Alessandro Biondi. He was born in Palermo, where he obtained his Master's degree in Computer Engineering at the University of Palermo, with an experimental thesis on developing security systems for instant messaging services for Italtel Ltd., an Italian ICT company. In 2015 he participated in the Erasmus+ project at The University of Technology of Belfort-Montbéliard (UTBM). His research areas concern the design and implementation of embedded and cyber-physical systems, focusing on security. Since March 2019, he has been working on a research project with the Rete Ferroviaria Italiana (RFI) to create a kernel for railway safety-critical systems.

FEDERICO AROMOLO is a Ph.D. student at the Real-Time Systems Laboratory (ReTiS Lab) of the Scuola Superiore Sant'Anna (Pisa, Italy). He holds a Master of Science degree in Embedded Computing Systems from the Scuola Superiore Sant'Anna and the University of Pisa. His research interests are in the area of real-time embedded systems and include real-time scheduling and synchronization algorithms, design and implementation of embedded and cyber-physical systems, real-time operating systems, and advanced robotics and artificial intelligence applications.

PIETRO FARA is a Ph.D. student since October 2019 at the ReTiS Laboratory of Scuola Superiore Sant'Anna – Pisa (Italy). In November 2018 he started to work at the ReTiS Lab as a research fellow in a project in partnership with Rete Ferroviaria Italiana (RFI) with the objective to design and develop a real-time kernel for safety-critical systems. He achieved the Master degree in Computer Engineering at the University of Pisa In 2015.

GABRIELE SERRA (Member, IEEE) is a Ph.D. student at Scuola Superiore Sant'Anna, situated in Pisa, Italy. He proudly works in the Real-Time System Laboratory (ReTiS) under the supervision of Prof. Giorgio Buttazzo. He received an M.Sc. in Embedded Computing Systems, a course offered jointly by Università di Pisa & Scuola Superiore Sant'Anna. In May 2019, he started working on an ongoing project in partnership with Rete Ferroviaria Italiana targeting the design of a real-time operating system to be used in the railway scenario.

DANIEL CASINI (Member, IEEE) Daniel Casini is Assistant Professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. He graduated (cum laude) in Embedded Computing Systems Engineering, a Master degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna of Pisa (with honors), working under the supervision of Prof. Alessandro Biondi and Prof. Giorgio Buttazzo. In 2019, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include software predictability in multi-processor systems, schedulability analysis, synchronization protocols, and the design and implementation of real-time operating systems and hypervisors.

ALESSANDRO BIONDI (Member, IEEE) is Associate Professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. He graduated (cum laude) in Computer Engineering at the University of Pisa, Italy, within the excellence program, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

GIORGIO BUTTAZZO (Fellow, IEEE) is full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa, received a M.S. degree in Computer Science at the University of Pennsylvania, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He is Editor-in-Chief of Real-Time Systems, Associate Editor of the ACM Transactions on Cyber-Physical Systems, and IEEE fellow since 2012. He has authored 7 books on real-time systems and more than 300 papers in the field of real-time systems, robotics, and neural networks, receiving 11 best paper awards.