An Integrated Real-Time and Security Scheduling Framework for CPS

Kriti Kansal

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Thidapat Chantem, Chair

Yue Wang

Chang Woo Min

April 27, 2023

Arlington, Virginia

Keywords: Real-time systems, Security, Scheduling, Resilience

An Integrated Real-Time and Security Scheduling Framework for CPS

Kriti Kansal

(ABSTRACT)

In the world of real-time systems (RTS), security has often been overlooked in the design process. However, with the emergence of the Internet of Things and Cyber-Physical Systems, RTS are now frequently used in interconnected applications where data is shared regularly. Unfortunately, this increased connectivity has also led to a larger attack surface. As a result, it is crucial to redesign RTS to not only meet real-time requirements but also to be resilient to threats. To address this issue, we propose a new real-time security co-design task model, and an accompanying scheduling framework, where schedulability can be used to indicate whether both real-time and security requirements are met. Our algorithm is designed to be flexible, allowing different security mechanisms to be used along with real-time tasks. Specifically, we augment the frame-based task model by introducing an n-dimensional security matrix, which serves as a powerful tool to enable our approach. This matrix clearly indicates which defense mechanisms are available for each task in the system by storing the worst-case execution times of tasks. Then, we transform the problem of maximizing security, subject to schedulability, into a variant of the knapsack problem. To make this approach more practical, we implement a fully polynomial time approximation scheme (FPTAS) that reduces the time complexity of solving the knapsack problem from a pseudo-polynomial to a fully polynomial. We also experiment with a greedy-heuristic approach and compare the results of both algorithms.

An Integrated Real-Time and Security Scheduling Framework for CPS

Kriti Kansal

(GENERAL AUDIENCE ABSTRACT)

Real-time systems are computer systems that need to respond to events in a timely manner. In the past, these systems were designed without much consideration for security. However, with the increasing use of interconnected devices and systems, it has become important to make sure that real-time systems are secure and protected against malicious attacks. To address this issue, we propose a new approach for designing real-time systems that prioritizes security from the very beginning. Our approach allows for different security tasks to be executed depending on the system's needs, and we use a two-dimensional security matrix to help with this. We also introduce a way to solve the security problem that is faster and more efficient than previous methods. Our experimental results show that our new approach significantly reduces the time and effort required to solve the security problem while still producing good results.

Dedication

To mom and dad, Akshat, and Varun.

Acknowledgments

I would first like to thank my advisor, Dr. Thidapat Chantem. Without her guidance, encouragement, constructive criticism, and invaluable advice, none of this work would have been possible. I would also like to thank the other members of my advisory committee, Dr.Chang Woo Min for providing insights into the security implications of the work presented here, and Dr. Joseph Wang for comments and advice which have helped mold this work into its current final form. No research is done in a vacuum, especially in this work. I would like to thank Dr. Nathan Fisher and Dr. Sanjoy Baruah. Their ideas and work inspired the research presented here. I would also like to thank Pratham, Tanmaya, Rajarshi, and Radhika for their help. I am especially grateful to Varun for his constant support, and encouragement and for being a source of happiness throughout the years. Finally, I would like to thank my parents, and my brother who have kept me motivated, supported, and helped me regardless of the situation. Without their love, hard work, and sacrifices, none of this would have ever been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time systems (RTS) conform to two main properties: logical correctness and timeliness. Real-time computing systems must react to the events in the environment by strictly following certain timing constraints. The result of a late reaction is useless and in some cases can also be fatal. Today, several complex real-life applications rely highly on real-time systems. [4]

RTS are now more interconnected than ever before. Technological advancements and data sharing have allowed these systems to be used in cyber-physical systems (CPS) and Internet of Things (IoT) applications to improve the quality of our life by making smart on-time decisions. This increased connectivity, however, does come with its own challenges: RTS now have a larger attack surface than they did before. This provides a malicious party with additional entry points to potentially infiltrate and modify critical operations of a real-time system that can directly affect safety and security. For example, a surveillance unmanned aerial vehicle (UAV) may be used to collect sensitive information such as coordinates of interest. An attacker can hack into the system and extract information from the UAV or introduce a fault so that the UAV might malfunction and crash in a civilian neighborhood. There have been several high-profile attacks on CPS such as on manufacturing industries, automobiles, and medical devices [5, 6, 12]. For example, Stuxnet [9] is a computer worm that is built to attack supervisory control and data acquisition (SCADA) systems that are extensively used in manufacturing industries. Stuxnet was used to infect the nuclear plant

and caused significant damage to Iran's nuclear program. In such cases, it is clear that we must protect these systems.

In this paper, we propose a more general, integrated security/ real-time scheduling framework that can support different types of security mechanisms such as encryption, isolation, and control-flow integrity checks. Our approach is unique, such that in our framework, schedulability analysis can provide **both** timeliness and security guarantees. We consider frame-based tasks where each task is provided with an n-dimensional security matrix that denotes the additional worst-case execution times associated with the different defense mechanisms that are available to that task. For each defense mechanism, different levels of protection may also be available. These defense mechanisms can be of any type — cache flush to prevent information leakage, encryption to secure data acquisition, Trusted Execution Environment (TEE) for isolation to prevent unauthorized access, control-flow integrity (CFI) to mitigate software-related security attacks, etc. The goal is to maximize security by selecting appropriate entries in the security matrix such that the task set is schedulable.

This paper claims three major **contributions**:

1. We *propose a security-integrated real-time system task model* for frame-based periodic task sets. The model offers a range of security mechanisms, enabling the task set to select the ones that best suit their requirements.

2. Based on the task model above, we *propose a scheduling framework* that selects the most appropriate security mechanism for each task. The main objective of this framework is to achieve maximum security without compromising real-time properties.

3. Our framework is implemented in two distinct ways: a knapsack-based approach and a greedy-heuristic method. We *validate and compare* both algorithms on a set of 90,000 randomly generated task sets.

Chapter 2

Review of Literature

It is crucial to secure time-sensitive CPS by design. While some research has been conducted on detecting attacks in real-time systems, security is considered separately from timeliness. For example, some attack detection algorithms focus on lightweight intrusion detection and cryptography [10, 20, 23] but do not consider real-time properties. Some papers have designed general software countermeasures, that are not algorithm specific and can be employed at the kernel and/or compiler level [1].

There is work that discusses designing a secure real-time system from the ground up. These papers however may not be broadly applicable since they make architectural changes. [14, 25]. Some papers opportunistically monitor the security needs of the system, depending upon the available dynamic slack. However, this monitoring is done on additional hardware that is sometimes difficult to retrofit into legacy systems.

On the other hand, some researchers have focused on purely software approaches. These algorithms however only focus on providing one type of security. For example, in [21, 22], a security-aware scheduler was developed to address issues of information leakage through shared caches. This scheduler was designed to ensure that the cache is flushed after a highly-sensitive task finishes execution, effectively mitigating the risk of data breaches caused by shared caches. In the research paper [26], a team of researchers developed an ingenious schedule randomization protocol known as the "task shuffler." By randomizing the execution of tasks within a given task set, this protocol effectively eliminates the risk of timing

interference attacks, wherein an attacker attempts to reconstruct the task schedule based on prior system behavior. The task shuffler enhances system security and safeguards against timing attacks.

Lastly, some researchers aim to design secure real-time systems by allowing the systems to dynamically choose the appropriate level of security that can be added based on real-time parameters. This approach involves assigning a “rank” to each security mechanism available, giving the system the ability to dynamically select the most appropriate defense mechanism for each situation. The paper [24] outlines a selection of security services for users to choose from. To assist in this decision-making process, the authors assign a ”benefit-cost ratio” to each security service and evaluate the available slack to determine the most appropriate security task. Essentially, this method involves a form of greedy security assignment, wherein the highest benefit-cost ratio tasks are prioritized.

The paper [13] proposes the creation of ”security groups” to address the diverse security needs of various tasks. Rather than executing security mechanisms on the entire task, the authors suggest applying security measures only to the number of data items (i.e., the length of the message) requiring protection within each task. To aid in scheduling decisions, the authors utilize the Integer Linear Programming Technique.

Research papers such as [16, 17] propose the use of encryption as the primary defense mechanism against security threats, albeit with varying security levels. These papers employ feedback control theory to determine appropriate security measures based on CPU utilization and energy consumption both. Additionally, tasks are labeled based on their vulnerability to outside attacks, with some papers also factoring in the deadline miss ratio into the feedback loop. However, in the paper [15], a two-level feedback loop is employed that is controlled based on the running time status of tasks, offering a novel approach to enhancing system security.

In [28] authors provide only encryption is provided as a defense mechanism with varying number of rounds. To determine the optimal number of rounds to assign to each task, the authors utilize dynamic programming. By making optimal decisions and considering various factors, they are able to provide a solution in polynomial time, resulting in a more efficient and effective allocation of encryption resources.

In practice, different tasks may require varying degrees of protection, and thus, different security mechanisms.

Chapter 3

Attack Model, System Model, and Problem Statement

3.1 Attack Model

Real-time systems (RTS) may encounter different types of threats based on the specific system and the motives of the attacker. One such instance is when attackers tamper with, monitor or change the messages exchanged between system components. They may also alter the way sensor inputs are processed, manipulate actuator commands or attempt to modify the control flow of the system. In addition, instead of attempting to forcefully crash the system, an intruder who is gathering information may choose to monitor the system's behavior and collect data for future use. This could involve using side channels to observe the system's operations and deduce valuable information such as hardware and software architecture, user tasks, and thermal profiles. Such information could then be utilized to plan a more effective attack later on.

Although attacks can be broadly categorized, our focus in this work is not on any one specific attack. Instead, our goal is to provide flexibility, allowing the system designer to choose the most suitable defense mechanisms based on the type of system and potential attacks. The security mechanisms can be classified as protection, detection or response, and are chosen based on the specific needs of the system. Take, for instance, an autonomous car - a cyber-

physical system comprising several interconnected real-time system components. Suppose one of the RTS components is responsible for gathering sensitive data that could prove catastrophic if accessed by an attacker. In such a scenario, it is likely that the attacker may try to obtain sensitive information through side channels. Hence, the system designer must select appropriate security mechanisms. For instance, measures such as intrusion detection or encryption could be employed to safeguard sensitive data.

Our emphasis is on the fact that instead of creating particular defense mechanisms that solely target specific attack behaviors, the generic framework proposed in our work empowers the system designer to incorporate security mechanisms of the appropriate type as and when required.

3.2 Proposed Real-Time Task Model

We consider a uniprocessor system consisting of n independent frame-based real-time tasks $\Gamma = \{\tau_1, \tau_2, .., \tau_N\}$. Each task $\tau_i \in \Gamma$ is characterised by:

$$\tau_i = \{c_i, T, \pi_i\} \tag{3.1}$$

where $c_i \in \mathbb{R}^+$ is the worst-case execution time (WCET) of the task alone without any security, $T \in \mathbb{R}^+$ is the task period that is the same for all tasks, and $\pi_i \in \mathbb{N}$ is the priority associated with the task (with the interpretation that tasks with larger values of this parameter have priority over those with smaller values.).

Since existing real-time task models such as the one above are usually limited to expressing only the timing parameters of a real-time task, it is necessary to consider the security aspects of the real-time task separately. We propose modifying the above tuple in Equation (3.1)

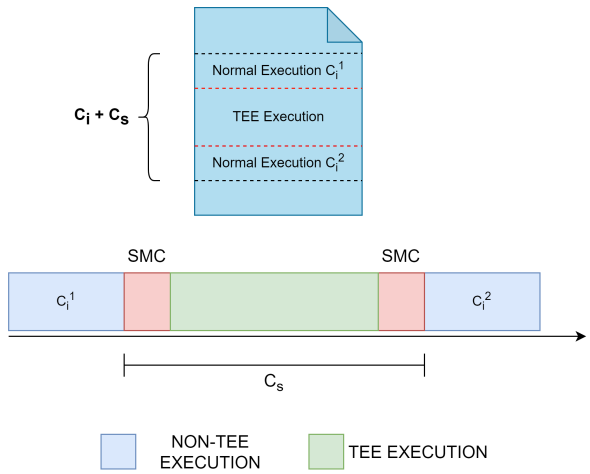

Figure 3.1: An Example Timeline For a Task When Executed With TEE

to include the security aspects of the task. For example, if we would like to execute a trusted execution environment (TEE) for our task, we would execute it in a way shown in Figure 3.1. This, however, is a very specific example and thus is very limiting on the "level" of security we can provide our tasks. To overcome this limitation and for the sake of facilitating schedulability and security analyses, we propose a highly expressive, general, integrated security/ real-time task model by modifying the above Equation (3.1). We define $\mathbf{S}_i$, an M-dimensional vector where M is the number of defense mechanisms that may be associated with the task τ_i. These defense mechanisms are then ranked based on their effectiveness against attacks. Then, $\mathbf{S}_i$ will allow us to schedule a diverse range of security mechanisms of different "levels." In other words, we augment the periodic task model tuple like this:

$$\tau_i = \{c_i, T, \pi_i, \mathbf{S}_i\} \tag{3.2}$$

The next obvious question here is: why do we need security mechanisms of different levels? There may be cases where a task set (along with their security mechanism counterparts) is not schedulable on a target platform because defense mechanisms take too long or there are insufficient computational resources to execute the workload. In many real-world scenarios,

a simpler defense mechanism may be acceptable in such a case. As an example, 128-bit encryption may be sufficient although 256-bit encryption would be preferred, and it is better to provide some defense rather than none. In addition, depending on the system state, it may be possible to rank the order of different security aspects in order of importance (e.g., encryption is desired more than isolation). We will assume that a ranking is provided for the tasks in the system and that for each aspect of security, a set of defense mechanisms is also given. For a given aspect of security, the defense mechanisms will have different time overheads and effectiveness. Again, these defense mechanisms are ranked.

Let us explore and understand the security parameters via an example. Consider our previous example of a UAV that is on its way to collect the coordinates of an enemy camp. In this scenario, we may wish to protect the data acquisition task to safeguard collected intelligence. Example applicable defense mechanisms available include but are not limited to TEE to provide isolation, encryption, and CFI. We can, for example, protect the task using encryption based on the amount of slack available on the processor. That means we could provide security mechanisms of varying protection levels. For example, we propose having three different levels of security – 0, 1, 2, and 3. Here, 0 signifies no protection, 3 is the highest level of protection, and 1 is the least. For this example, let us suppose that the security level (SL)-1 corresponds to TEECheck [18], and SL-2 corresponds to control-flow integrity (CFI) mechanisms [19]. Lastly, the highest level of security SL-3 can be the 256-bit encryption method. The worst-case execution times of these security tasks are stored in the vector $\mathbf{S}_i$, an M-dimensional vector. In this example, Since, we have the option to choose from three different security mechanisms, $\mathbf{S}_i$ is a 3-dimensional vector. It will look like this:

$$\mathbf{S} = \{C_{tee} \quad C_{cfi} \quad C_{encry}\} \tag{3.3}$$

The final WCET of task τ_i will be $C_i = c_i + c_s$, where c_s is the WCET of the security mechanism that will be selected from $\mathbf{S}_i$ in the Equation (3.3) above, according to an algorithm that we will discuss in the next two sections. For example, let us say, we have three tasks τ_1, τ_2, τ_3. The task attributes can be described as in Table 3.1, where π is the priority of the task (3 is the highest and 1 is the lowest). In this example, let us assume that the common period of all the tasks is 50 units. Furthermore, suppose that the individual vectors $\mathbf{S}_i$ for all tasks, when combined to represent the whole system, is as given in Equation (3.4). We will call this our "security matrix" of dimension $N \times M$, where N is the number of tasks in the system and M is the number of defense mechanisms available for each task. Therefore, the first row, i.e $S(1, m)$ where $m \in \{1, 2, ..., M\}$ represents the available security mechanisms for task-1 (τ_1).

For example, S(1,1) is the WCET of the security mechanism of security level-1 (SL1) for τ_1, i.e C_{tee}. Similarly, S(1,2) is the WCET of the security mechanism at SL2, i.e C_{cfi}. Subsequently, S(2,3) is the WCET of the security mechanism for τ_2 at SL3. In this example, the utilization of the task set in Table 3.1 is 60% and the additional utilization of security mechanisms is roughly in the range of 40-60%.

As an additional point, we can enhance our security matrix by introducing an extra dimension. Each security mechanism can now have multiple sub-levels of security. For instance, when a task necessitates encryption, we can offer three levels of C_{encry}: RAS 128-bit encryption, AES 128-bit encryption, and AES 256-bit encryption. With this approach, our security matrix would become a 3-dimensional structure with a size of $N \times M \times K$, where N represents the number of tasks, M represents the number of security mechanisms, and K represents the number of security levels available for each security mechanism. In this paper, we will use the two-dimensional security matrix of size $N \times M$ for simplicity but our approach can be readily extended to the three-dimensional case.

Table 3.1: Task Attributes.

Hyper period = 50 units
Util = 60%

Task	C	T	π
τ_1	14	50	3
τ_2	10	50	2
τ_3	6	50	1

$$\mathbf{S} = \begin{bmatrix} C_{tee} & C_{cfi} & C_{encry} \\ C_{tee} & C_{cfi} & C_{encry} \\ C_{tee} & C_{cfi} & C_{encry} \end{bmatrix} = \begin{bmatrix} 6 & 8 & 16 \\ 2 & 14 & 15 \\ 2 & 10 & 18 \end{bmatrix} \tag{3.4}$$

3.3 Problem Statement

Consider a set of frame-based tasks characterized by time period T. Each task is associated with a worst-case execution time c_i, and a priority π_i. The tasks are also provided with available security mechanisms in the form of an $N \times M$ security matrix $\mathbf{S}$, as discussed above. Our objective is to choose the most suitable security mechanisms from the given security matrix, which not only provide protection against malicious parties but also comply with real-time requirements. In other words, the security mechanisms will be chosen based on the available free time on the processor. This means we make use of the slack in the system. Since this is a frame-based task model, the slack can be computed in a straightforward manner by simply subtracting the total execution time of all tasks from the system's time period T. Equation (3.5) below shows the calculation of slack (T').

$$T' = T - \sum_{i=1}^{n} C_i \tag{3.5}$$

With this, we can ensure that our system remains secure and reliable while maintaining the expected level of performance. Our objective is to maximize the security of the system during the time period T, as described by Equation (3.6). Here, SL_i is the security level of the chosen security mechanism for task τ_i. An overview of our methodology is given in Figure (3.2). First, we define a new security real-time co-design task model for frame-based task sets. Second, using this task model we design an algorithm that selects appropriate security mechanisms for these tasks while taking into account their real-time properties.

$$\max \sum_{i=1}^{N} SL_i \tag{3.6}$$

The security level of the "system" or the "task set" can be quantified as the sum of the chosen security levels for all individual tasks: $\sum_{i=1}^{N} SL_i$. It is important to note that the system is considered to be more secure when the security level of the system is maximized. This is because high security levels mean that we have chosen defense mechanisms with higher effectiveness against attacks. Specifically, the purpose is to increase the security of the system as much as possible, subject to the availability of free space (or slack) on the processor.

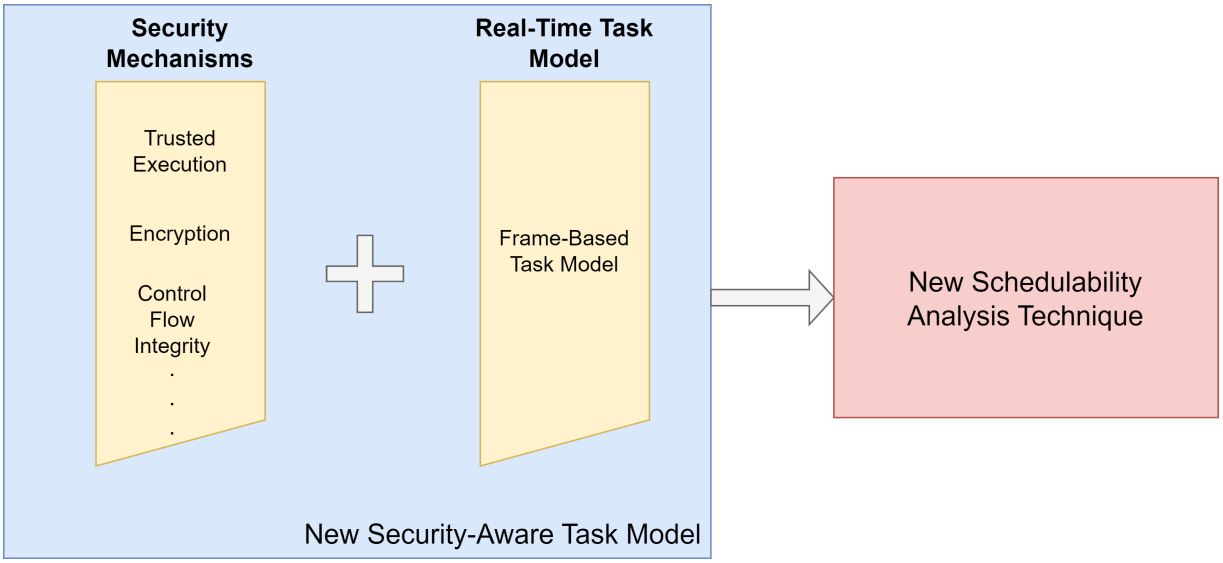

Figure 3.2: Proposed Real-Time/Secure Task Model And Framework Overview

Chapter 4

Approach

This section will make use of the aforementioned task model to outline two algorithms that are designed to maximize the security of the system. As an example, let us consider a task set consisting of three tasks: τ_1, τ_2, and τ_3. These tasks are characterized by their WCET C_i, time period T, and priority π_i, as shown in Table 3.1. Additionally, we will reuse the matrix of available security mechanisms (Equation (3.4)). As stated earlier, each security mechanism is associated with one or more "security levels," which reflects its effectiveness in providing defense against potential threats. This attribute allows us to quantify the level of protection offered by each mechanism and make informed decisions regarding their selection for a particular task or system. To achieve this, we explore two different approaches.

4.1 Knapsack-based Approach

We now describe the problem transformation from calculating the maximum security level of the system to a variant of the knapsack problem. This section provides context and uses the task attributes from Table 3.1 and security matrix $\mathbf{S}$ from Equation (3.4) as an example to showcase the algorithm.

In the traditional knapsack problem, one is given a set of items such that each item has a corresponding profit and weight, and a knapsack of specified weight capacity. The aim of the problem is to find a subset of the set of items such that the total profit is maximized and

the aggregate weight of the selected items is less than or equal to the maximum allowable capacity of the knapsack. Our aim is to determine the maximum possible security level for the system, represented by the sum of the security levels of the selected mechanisms, subject to schedulability. In our approach, we treat each job as an item and assign two attributes to it: weight and profit. The weight of each item corresponds to the worst-case execution time of the available security mechanisms associated with that job. Notably, the weight values are organized into a 2-dimensional matrix, where the "weight" attribute is classified into N classes, each containing M_i items. In this context, the number of classes represents the number of tasks in the system i.e. N, while M_i denotes the number of available security levels for a given security mechanism for task τ_i. The second attribute "profit" is also represented as a 2-dimensional matrix where each value represents the security level of the available security mechanism. In short, each item j $\in M_i$ has a profit p_{ij} and a weight w_{ij}. In general, the matrices **w** and **p** will look like this:

$$\mathbf{w} = \begin{bmatrix} C_{tee} & C_{cfi} & C_{encry} \\ C_{tee} & C_{cfi} & C_{encry} \\ C_{tee} & C_{cfi} & C_{encry} \end{bmatrix} \tag{4.1}$$

$$\mathbf{p} = \begin{bmatrix} SL1 & SL2 & SL3 \\ SL1 & SL2 & SL3 \\ SL1 & SL2 & SL3 \end{bmatrix} \tag{4.2}$$

Using **S** from Equation (3.4) the profit and weight matrix for the example task set in Table 3.1 can be transformed into Equations (4.3) and (4.4) given below.

$$\mathbf{w} = \begin{bmatrix} 6 & 8 & 16 \\ 2 & 14 & 15 \\ 2 & 10 & 18 \end{bmatrix} \tag{4.3}$$

$$\mathbf{p} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \tag{4.4}$$

Having established the profit and weight matrices for our jobs, we turn our attention to the third attribute required for the knapsack problem — the maximum capacity c of the knapsack. In our case, the maximum capacity of the knapsack corresponds to the total available slack time in the system. This slack value is computed using Equation (3.5). The goal, then, is to maximize the sum of security levels (i.e., profit) such that the sum of execution times of the chosen security mechanisms (i.e., weights) is less than or equal to the slack available (i.e., the capacity of knapsack).

Note that while we aim to maximize security by selecting the highest security level for each task (Equation (3.6)), another objective function can readily be used with our proposed task model as long as its relationship with the profit (discussed more below) can be defined. Also, although we implicitly assume that all tasks are equally important, weights can be incorporated into the objective function to prioritize defense for the more important tasks.

A 0-1 Multiple-Choice Knapsack Problem (MCKP) [11] is a variant of the knapsack problem. In 0-1 MCKP, each item has multiple options or "choices" of weight and value, i.e multiple classes. The decision-maker must select one option per item, and the total weight of the selected options must not exceed the capacity of the knapsack. The key difference between the knapsack problem and MCKP is that in the knapsack problem, each item has only one

weight and value, while in MCKP, each item has multiple weights and values as shown in Equations (4.1) and (4.2). Therefore, our optimization problem can be transformed into a 0-1 MCKP. Figure (4.1) provides a visual example representation of our problem as an MCKP. The figure shows that for every task, we have security mechanisms available of different "weights" and "profits". Our aim is to select the appropriate security mechanisms based on the slack available (i.e. the capacity of the knapsack).

Pisinger et al. in their book "Knapsack Problems" [11] formally define the 0-1 Multiple-Choice Knapsack problem as follows. Consider N mutually disjoint classes (i.e., number of tasks) each with M_i number of items (i.e., number of security levels for a given security mechanism) to be packed into a knapsack of capacity c (i.e., available slack). Each item j $\in$ M_i has a profit p_{ij} and a weight w_{ij}. Then the goal is to pick at most one item from each class such that the sum of profits is maximized and the corresponding sum of weights does not exceed the capacity c. We also introduce a binary variable x_{ij}. The value of x_{ij} is 1 if and only if item j is chosen from class N_i. The MCKP problem is formulated as:

$$\begin{aligned}
\text{maximize} \sum_{i=1}^{N} \sum_{j \in M_i} p_{ij} x_{ij} \\
\text{subject to} \sum_{i=1}^{N} \sum_{j \in M_i} w_{ij} x_{ij} \leq c \\
\sum_{j \in M_i} x_{ij} = 1, \quad i = 1, \ldots, N, \\
x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, N, \quad j = 1, \ldots, M_i
\end{aligned} \tag{4.5}$$

The knapsack problem has been well-studied in mathematical programming. In most cases, this maximization problem is solved using dynamic programming (DP). However, dynamic programming solutions are often both computationally and memory intensive. It is well known that the knapsack problem is an example of an NP-hard optimization problem and

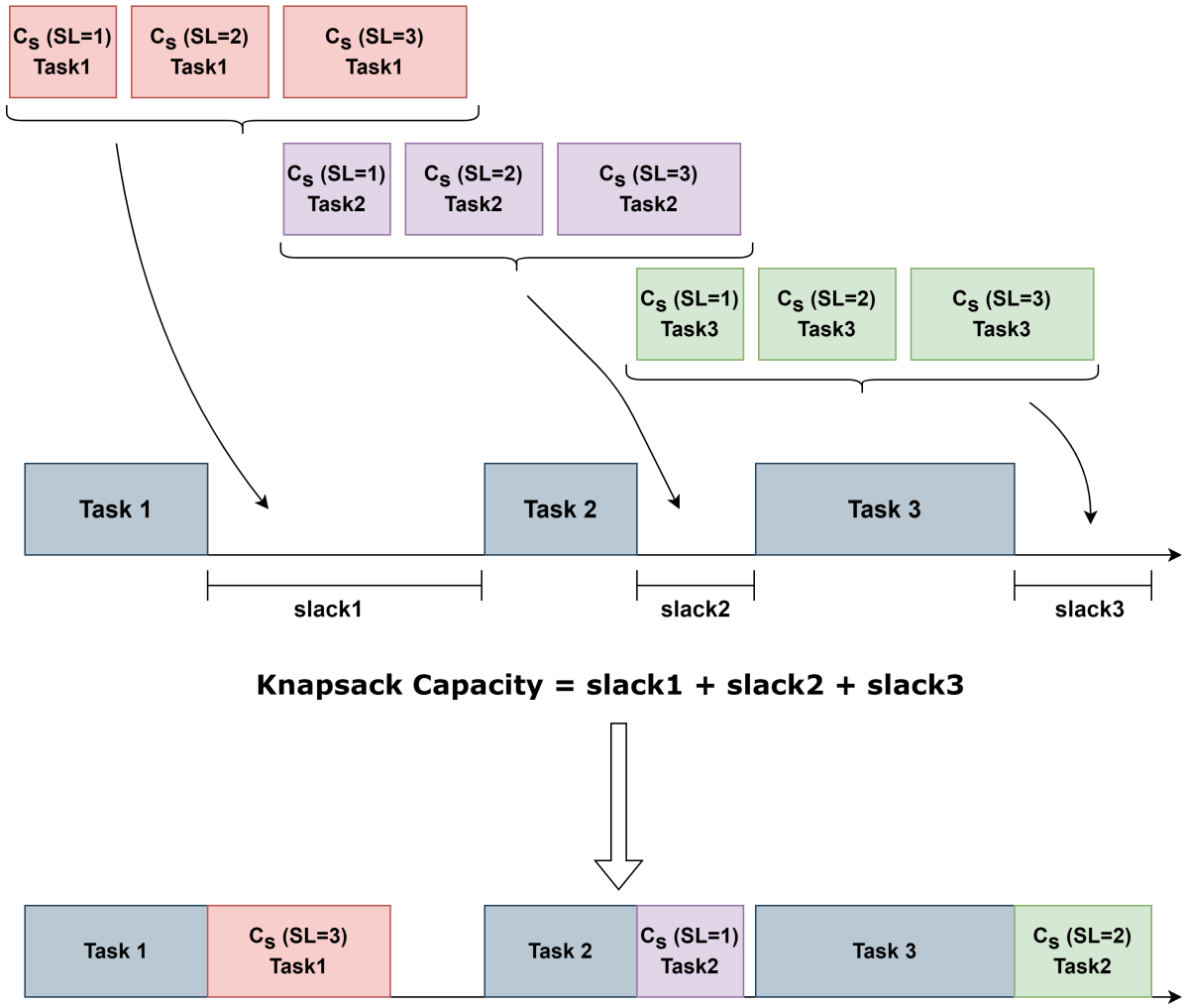

Figure 4.1: An Example that Showcases the Multiple-Choice Knapsack Problem

the dynamic programming approach solves the problem in pseudo-polynomial time [11]. However, algorithms known as approximation schemes are a viable option that will help us solve the problem in polynomial time. In this paper, we implement the Fully-Polynomial Time Approximation Scheme (FPTAS) described by Bansal et al. in [2].

4.1.1 Full-Polynomial Time Approximation Scheme

Let us formally define an approximation scheme. Let Π be an optimization problem with the objective function f_Π, an optimal solution S^*, and an optimal value of OPT(Π). Then for input (I; ϵ), where I is an instance of Π and the error parameter is $\epsilon > 0$, algorithm **Z** is known as an approximation scheme for Π if **Z** outputs a solution S_Z that satisfies the following inequality for a maximization problem:

$$f_\Pi(I; S_Z) \geq (1 - \epsilon)OPT(\Pi) \tag{4.6}$$

Furthermore, **Z** is a *fully-polynomial time approximation scheme* or FPTAS if for input (I, ϵ) the running time complexity of **Z** is a polynomial function in both the size of I and $\left(\frac{1}{\epsilon}\right)$. Therefore, with an FPTAS algorithm, we can find a near-optimal solution in polynomial time. The optimality of the approximate answer can be controlled via the error parameter ϵ. Ideally, we keep ϵ values close to 0, such that the approximation introduced in the FPTAS algorithm is as low as possible. The pseudo-code for the algorithm is summarised in Algorithms (1) and (2). The algorithm takes the following inputs: **p**, **w**, ϵ, and knapsack capacity c. The output of the algorithm will be the maximum sum of profits such that the weight is less than or equal to the knapsack capacity. There are three main steps in the algorithm:

Algorithm 1 Dyer-Zemel Algorithm

```
procedure FIND_UPPERBOUND
    while true do
        W ← 0          P ← 0
        for all class N_i in N do                                  ▷ Step 1(a): Make Pairs
            Make pairs (ij_1, ij_2) such that (w_{ij_1} ≤ w_{ij_2})
            if item j_1 dominates item j_2 then
                Delete item j_2 from class N_i and pair j_1 with another item
            end if
        end for
        for all class N_i in N do
            if M_i = 1 then
                W ← W + w_ij    P ← P + p_ij
                Fathom class N_i
            end if
        end for
        for all pairs(ij_1, ij_2) do
            Calculate slope γ_{ij_1 ij_2} = (p_{ij_1} − p_{ij_2}) / (w_{ij_1} − w_{ij_2})
        end for
        Find median γ of all slopes {γ_{ij_1 ij_2}}
        for all class N_i in N do                                  ▷ Step 1(b): Calculate M_i(γ), φ_i, ψ_i
            M_i(γ) = arg max_{j∈N_i} {p_ij − γ w_ij}
            φ_i = arg min_{j∈M_i(γ)} {w_ij}
            ψ_i = arg max_{j∈M_i(γ)} {w_ij}
        end for
        if W + Σ_{i=1}^N w_{iφ_i} ≤ c < W + Σ_{i=1}^N w_{iψ_i} then   ▷ Step 1(c): γ is optimal
            W ← W + Σ_{i=1}^N w_{iφ_i}
            P ← P + Σ_{i=1}^N w_{iφ_i}
            return z* = P + (c − W)γ
        end if
        if Σ_{i=1}^N w_{iφ_i} ≥ c then                             ▷ Step 1(d): Determine items to be deleted
            for all pairs(ij_1, ij_2) do
                if γ_{ij_1 ij_2} ≤ γ then Delete item j_2 from N_i
                end if
            end for
        end if
        if Σ_{i=1}^N w_{iψ_i} < c then
            for all pairs(ij_1, ij_2) do
                if γ_{ij_1 ij_2} ≥ γ then Delete item j_1 from N_i
                end if
            end for
        end if
    end while
end procedure
```

Step 1: Calculate Upper Bound. First, let P^* be the optimal profit, then we find the lower and upper bounds on the optimal solution such that $P_0 \leq P^* \leq 3P_0$. This can be done with the help of the Dyer [8] and Zemel [27] algorithm that runs in O(n) time. We note that although the bound is obtained from linear programming theory, we do not require an LP solver to calculate P_0; rather, P_0 can be determined in linear time using the median-find algorithm (as discussed in the description of the Dyer-Zemel algorithm).

Algorithm (1) presents the pseudo-code for this step. In lines (5-17), we first make item pairs for each class separately such that $(w_{ij_1} \leq w_{ij_2})$. After this, we check if any classes can be fathomed, i.e., if they can be eliminated from further consideration. Once we have our item pairs, we move on to lines (20-29), where we determine four key values: the median slope (γ), $M_i(\gamma)$, ϕ_i, and ψ_i. Next, in lines (31-35), we check whether the median slope is optimal or not. If it is indeed optimal, then we have found our upper bound, and we exit the algorithm. However, if the median slope is not optimal, then we move on to lines (37-45), where we determine which item pairs need to be discarded before starting the loop again.

Step 2: Scale Profits. We now have the lower bound (P_0) and the upper bound ($3P_0$). We also have the error parameter ϵ. To make the computation in the next step more efficient, we will reduce the size of the search space. This can be achieved by scaling our profits. The new scaled profits $q_{ij} = \lfloor p_{ij}/K \rfloor$, where $K = \epsilon P_0/N$. This step is performed in lines (3-6) in the pseudo-code given in Algorithm (2).

Step 3: Dynamic Programming in Profits. Now that we have our upper bound ($3P_0$) and the scaled profits we will proceed to define the dynamic programming formulation in profits q_{ij}. For each class (i.e. each task in the task set) $i \in \{1, \ldots, N\}$ and profit (i.e. the security levels of security mechanisms) $p \in \{0, \ldots, 3P_0\}$, let S_{ip} be the set of items from classes $(1, \ldots, i)$ such that at most one item is selected from each class and the total profit

of the set is p. If no such set exists, we set S_{ip} to infinity. Then, we define $F(i, p)$ which holds the sum of the weights of all items in set S_{ip}.

Let us understand this with the help of our task set from Table 3.1 and security matrix Equation (3.4). For example, in the case of $i = 2$, and $p = 4$, we want to determine the value of S_{24} and subsequently $F(2, 4)$. In other words, we want to find the set of items in classes $(1, 2)$ such that the sum of their profit values is exactly $p = 4$. Looking at the values of the profit in Equation (4.4), we see that the sum of items with profits p_{12} and p_{22} is exactly $p = 4$. Therefore, $S_{24} = \{x_{12}, x_{22}\}$ and the value of $F(2, 4)$ is the sum of weights of these two items. Therefore, $F(2, 4) = 22$ because $w_{12} = 8$ and $w_{22} = 14$.

Additionally, We assume that the sum of items to produce a profit of 0 is 0. Therefore, $F(i, 0) = 0$. Also, for $i = 1$, $F(1, p)$ is known for every $p \in \{0, ..., 3P_0\}$. It is the weight of that one item in class which has minimum weight and exact profit p. While we were able to easily determine the value of $F(i, p)$ in the previous example, it becomes increasingly difficult to do so as the values of i and p become larger. Hence, to compute the remaining entries in the dynamic programming table, we rely on Equation (4.7) as given in Bansal et al. [2]. This step is shown in Algorithm (2) in lines (7-15). Finally, in lines (16-17), we check for the value of p for which $F(N, p)$ is less than or equal to knapsack capacity c for all $p \in \{3P_0, \ldots, 0\}$.

The approximation method for the Knapsack problem is applicable only if the scaling factor K is larger than 1. Therefore, FPTAS for Knapsack should be taken into account when $P > n/\epsilon$ [3]. In such cases, the dynamic programming algorithm may need substantial memory allocation for subproblem arrays, making the approximation method a preferred choice. It not only reduces running time but also considerably decreases memory usage. If weight and profit values are small $representedinfewbinarybits$, step #2 (scaling down profits) can be skipped, leading to a smaller search space for dynamic programming in step

#3. Our experiments in the next section use higher profit values.

$$
\begin{aligned}
F(1,p) &= \begin{cases} \min\{w_{ij}\}, \text{ where } p_{ij} = p, \\ \infty, \text{ otherwise.} \end{cases} \\
F(i,p) &= \begin{cases} \min\left\{F(i-1,p), \min\limits_{j \in N_i}\{w_{ij} \right. \\ \left. +F\left(i-1, p-p_{ij}\right)\}\right\}, \text{ when } p_{ij} \leq p, \\ F(i-1,p), \text{ otherwise.} \end{cases}
\end{aligned}
\tag{4.7}
$$

Algorithm 2 0-1 MCKP FPTAS

```
procedure MCKP_FPTAS
    P_0 ← FIND_UPPERBOUND()                          ▷ See Algorithm 1
    K ← ϵP_0/N
    for all p_ij do                                  ▷ Step 2: Scaling of Profits
        q_ij ← ⌊p_ij/K⌋
    end for
    for all i ∈ m do                                 ▷ Step 3: DP in Profits
        for all p ∈ 3P_0 do
            if i = 1 then
                Compute F(i, p) using Eq (4.7)
            else
                Compute F(i, p) using Eq (4.7)
            end if
        end for
    end for
    P* = max{p | F(N, p) ≤ c}
    return max(P_0, P*)
end procedure
```

4.2 Greedy-Heuristic Approach

The above algorithm provides a near-optimal solution to the optimization problem in polynomial time. Additionally, we propose a second approach, a greedy-heuristic algorithm,

which may or may not provide the optimal solution.

As the name suggests, the greedy-heuristic algorithm is designed to select security mechanisms in a greedy manner. The algorithm services the tasks one by one, as opposed to the knapsack approach where the security mechanisms are balanced across all tasks. Algorithm 3 shows the pseudo-code. In lines (3-4), the algorithm begins by selecting the defense mechanism that provides the highest effectiveness against threats for the first task, which is the ideal scenario as we aim to be as greedy as possible. We then check if the selected set of frames is schedulable (as shown in lines (7-9)), which is done by verifying if there is enough slack available to accommodate the chosen security mechanism. If the set is schedulable, we move on to the next task and again select the security mechanism with the highest effectiveness. If the set is not schedulable, we choose a security mechanism with lower effectiveness as shown in lines (11-12). We then repeat the process with the newly chosen mechanism and check for schedulability. The heuristic algorithm terminates if either of these conditions is met: if the available slack has been consumed by the chosen mechanisms, or if a security mechanism has been selected for all tasks in the task set.

Algorithm 3 Greedy-Heuristic Approach

```
procedure GREEDY(SL_MAX, total_slack)
    while not done do
        for all task τ_i in Γ do
            SL ← SL_MAX                                  ▷ Select maximum security
            while SL > 0 do
                update total WCET of τ_i
                if schedulable then                      ▷ Check if slack is enough
                    C_i ← c_i + c_SL
                    total_slack ← total_slack − c_SL
                    break
                else                                     ▷ Reduce security level and try again
                    SL ← SL − 1
                end if
            end while
        end for
    end while
end procedure
```

Chapter 5

Experiments and Results

In this section, we test and compare the two proposed algorithms. To validate the accuracy of the approximation scheme, we additionally compare the results of the approximation algorithm with the optimal outcomes obtained from the conventional dynamic programming approach. The experiments were performed using C++ on a quad-core processor (3.10 GHz) with 16 GB of RAM. Though the time taken by the algorithms may be platform dependent, the general trend showcases the relative performance of the proposed approaches against each other.

This comparison is performed over a set of 90,000 task sets. These synthetic task sets are generated using the UUnifast-Discard algorithm presented by Davis et al. in [7]. These task sets contain three tasks - τ_1, τ_2, and τ_3 which means $N = 3$. The task sets are frame-based, and schedulable with fixed priority scheduling algorithms. The task sets are generated for a hyper period of 50 units. The 90,000 task sets have been classified as follows. The first batch of 10,000 task sets has a utilization of 90%, which means that the amount of slack available is only 10%, which is 5 units. The second batch of 10,000 task sets has a utilization of 80% which means the amount of slack available is 10 units. Similarly, we have batches of 10,000 task sets with utilization 70%, 60%, 50%, 40%, 30%, 20%, and 10%.

In the next step, the add-on security task sets were generated using the UUnifast-Discard algorithm mentioned previously. Each security task set is a "security matrix" of dimension $N \times M$, where, N is the number of tasks in the system and M is the number of defense

mechanisms available for each task. In our experiments, $N = 3$ as stated above and $M = 3$ which means each task in the task set above has 3 security levels to choose from. We generate three batches of security matrices, each batch containing 10,000 matrices. In the first batch of the security matrix, we set the utilization of the picked security tasks to be around 20%. Similarly, the second batch contains matrices that will offer security tasks with a utilization of about 40%. The third and final batch will have a utilization of 60%.

Now that we have our task sets and security matrices ready, we test and run experiments on these data points that will showcase the similarities and differences, and pros and cons between the three different algorithms - MCKP (optimal DP algorithm), FPTAS, and Greedy-Heuristic.

In our first experiment, we run tests using the first batch of security matrices, i.e. the batch with 20% utilization. While our security matrix of 10,000 security task sets has a constant utilization, we vary the amount of slack present in the original task set. This way, we test the security matrix with 20% utilization on 90,000 task sets such that we experiment with all scenarios where the available slack time falls in the range {5, 10, 15, 20, 25, 30, 35, 40, 45 }. With an error parameter of $\epsilon = 0.001$, the results of this experiment are shown in Figure (5.1). In the figure, the x-axis represents the amount of slack available, whereas the y-axis demonstrates the maximum possible security level calculated for the system with all three algorithms. The security levels available to tasks are {100, 200, 300}. As previously noted, the security level values were increased to accommodate FPTAS, which performs better with larger input sizes. First, we notice that the traditional dynamic programming algorithm for MCKP results in the best security level of the system. However, it is already known that this solution is time-consuming and memory intensive. We also notice that the approximation scheme - FPTAS, produces a near-optimal solution. The results of FPTAS do not differ by more than $(1 - \epsilon)$ of the optimal solution. Secondly, the results from the greedy-heuristic

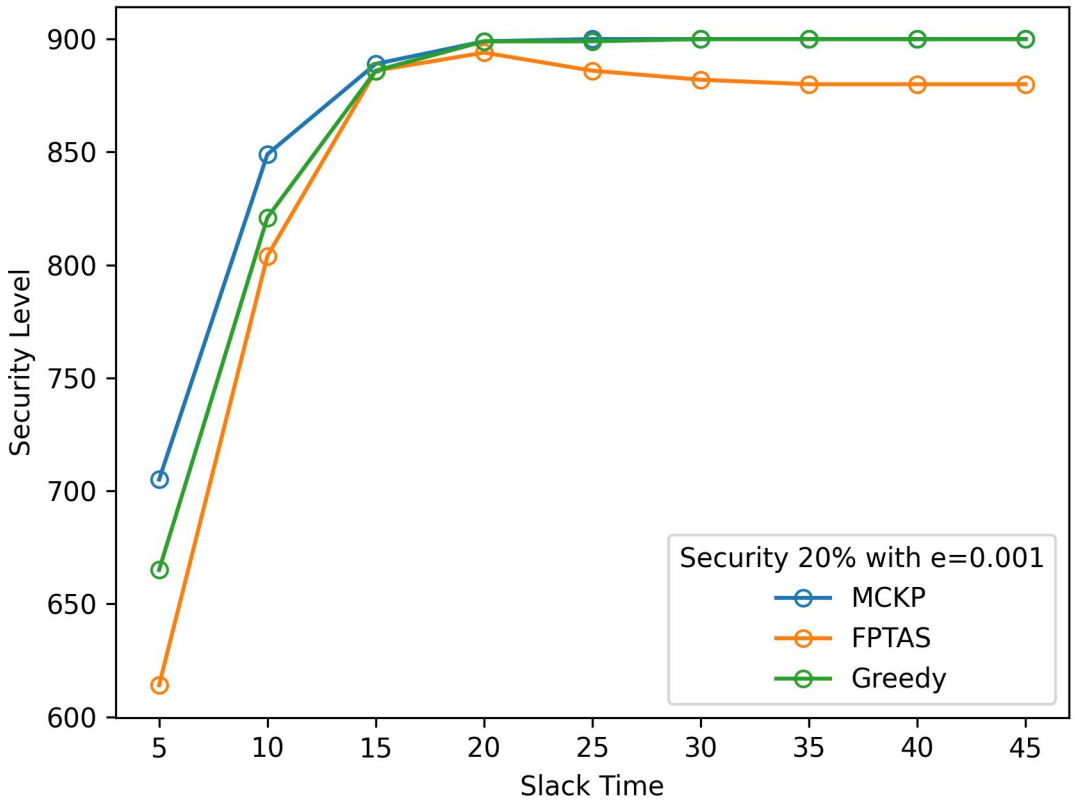

Figure 5.1: Fixed Utilization of Security Mechanisms at 20%

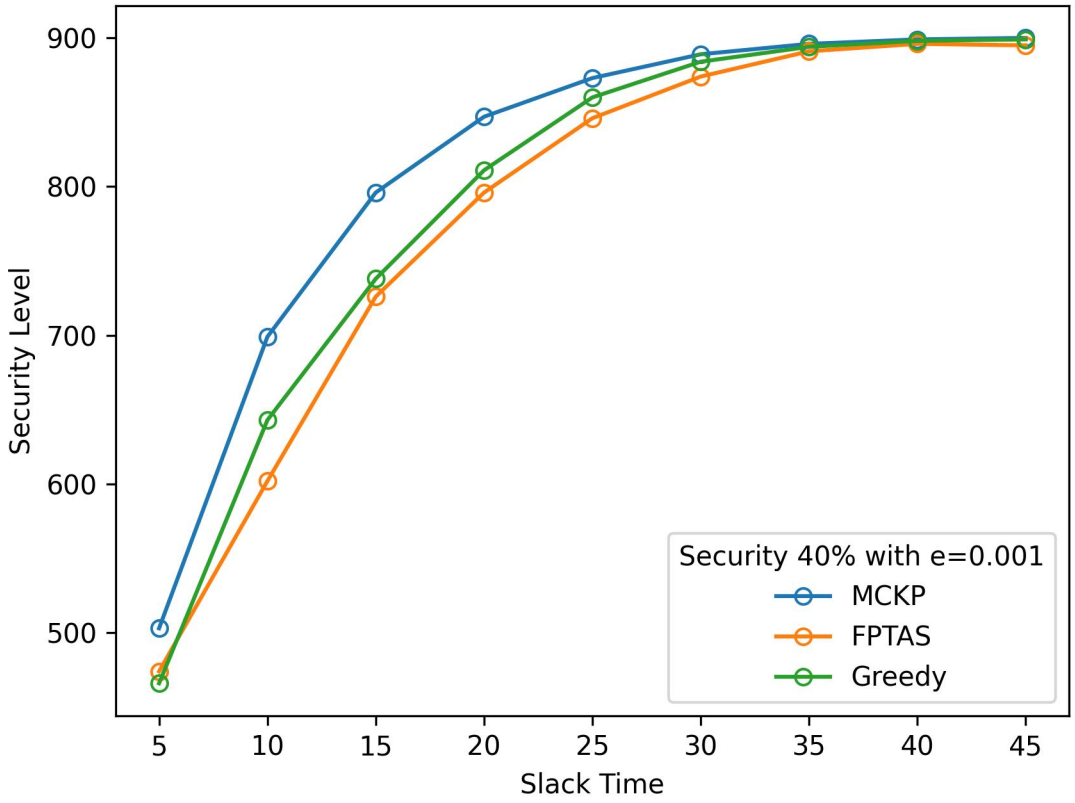

Figure 5.2: Fixed Utilization of Security Mechanisms at 40%

approach lie between those of the optimal algorithm and the approximation scheme. In one glance, we may come to the conclusion that the greedy-heuristic algorithm performs better than the FPTAS algorithm. However, that is not true. While FPTAS maintains a ”balance” of security levels in the task system, the greedy-heuristic approach greedily snatches security tasks of higher levels but does not guarantee that the other tasks in the system get any security. However, FPTAS as mentioned maintains a ”balance”, i.e it does not greedily assign security mechanisms but carefully distributes security mechanisms for all tasks in the original task set. This means that FPTAS will try harder to provide some, if not more, security to **all** tasks.

In our second experiment, we run similar experiments as stated above but this time with the second batch of security matrices. This batch contains matrices that have a utilization of about 40%. We observe similar results as above. The results are represented as a line graph in figure (5.2). Lastly, in our third experiment, we re-run the above experiments but with the third batch of security matrices, i.e., the batch with the utilization of 60%. The results observed are delineated in Figure (5.3).

One observation that we make by looking at the combination of all three Figures (5.1), (5.2), (5.3) is that there is a similarity between the three mentioned algorithms. We conclude that when the available slack in the system is very low, all three algorithms produce a low-security level for the system. This observation holds true because when the processor is mostly busy, i.e. we have less available slack, we can only assign security tasks of lower effectiveness. Subsequently, when the slack in the system is high, all three algorithms produce high-security levels for the system. This observation, again, going by the previous logic holds true. When the processor is mostly idle, the algorithms can select more effective security mechanisms for all tasks in the set.

When the system is mostly busy, or in other words when there is limited slack available, the utilization of security mechanisms becomes a critical factor. When the security utilization is only 20%, the security level achieved is at least 600. As we increase the utilization of security mechanisms, the security level decreases. For instance, with a security utilization of 60% and limited slack, the maximum achievable security level is only 400. This indicates that when resources are scarce, and security mechanisms have high utilization, all three algorithms will result in a lower overall security level for the system.

The 0-1 Multiple Choice Knapsack problem is an NP-hard optimization problem that can be solved using the conventional dynamic programming solution in pseudo-polynomial time with time complexity of O($c \sum_{i=1}^{N} M_i$), where c is the maximum allowed capacity of the knap-

sack, and M_i is the number of items in each class $i \in \{1, 2, ..., N\}$.It is worth noting that the time complexity of the 0-1 multiple choice knapsack problem is determined by the size of the input, and as the weight and profit values increase, the complexity also increases accordingly. Therefore, the conventional dynamic programming approach may become memory-intensive and contain large constants, making it less efficient for large inputs. In contrast, an approximation scheme provides the best trade-off for an NP-hard optimization problem in terms of performance guarantee and time complexity. The time complexity of the FPTAS, as discussed in previous sections, is O($\frac{nm}{\epsilon}$). In our experiments, we observed a similar pattern in the time complexity of FPTAS and DP as described by their mathematical representations. We found that FPTAS was able to process a batch of 10,000 task sets (each containing three tasks) 1.5 times faster than the DP solution. It is worth noting that the FPTAS implementation can be made more efficient by using better data structures in the code implementation of the FPTAS. Last, but not the least, the greedy-heuristic approach has a time complexity of O(nm). Although the greedy-heuristic solution is faster than the DP approach, it does not always provide an optimal solution.

5.1 Experimental Analysis with Varying Number of Tasks

Lastly, we conducted an experiment by varying the number of tasks in each task set. We generated 10,000 task sets, each containing a different number of tasks ranging from 3 to 8 (inclusive), and analyzed the results. The findings are illustrated in Figure (5.4). We observed that as the number of tasks increases, FPTAS performs exceptionally well, but the difference in outcomes between the DP approach and FPTAS is slightly more pronounced. Furthermore, the greedy approach performs as expected, always selecting the highest avail-

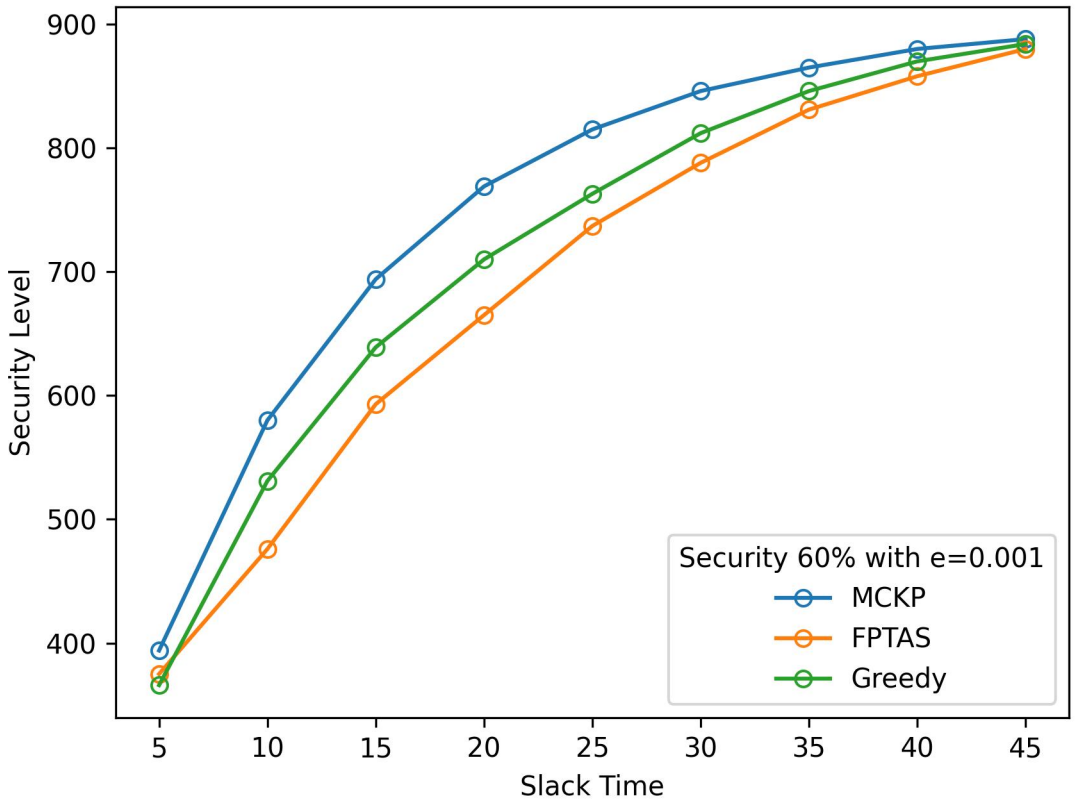

Figure 5.3: Fixed Utilization of Security Mechanisms at 60%

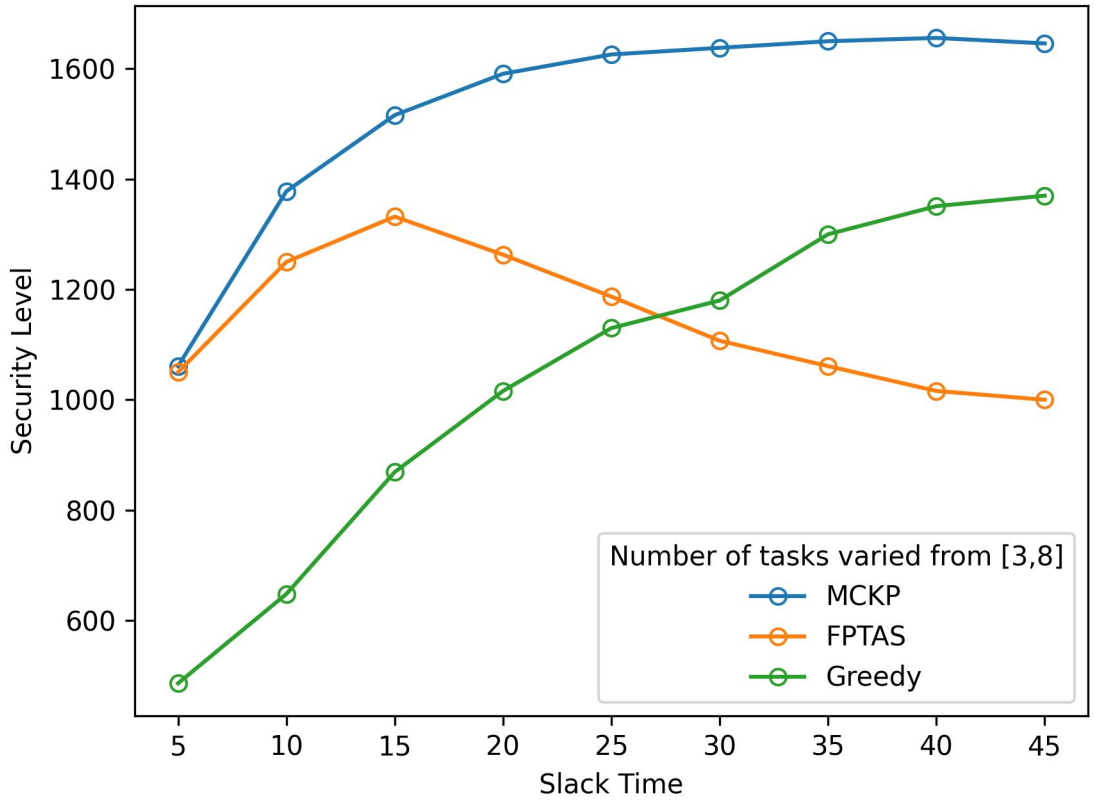

Figure 5.4: Number of Tasks Varied from [3,8] per Task Set

able security levels on a first-come-first-serve basis. As the available slack increases, the greedy algorithm's performance improves linearly.

To illustrate the impact of the number of tasks per task set, we conducted additional experiments with three batches of 10,000 task sets each. The first batch consists of three tasks per task set, the second batch contains five tasks per task set, and the third batch contains seven tasks per task set. The findings of this study are depicted in Figure (5.5). We observed that as the number of tasks increase, which corresponds to an increase in the number of classes in the knapsack problem, FPTAS generates results that deviate slightly from the expected outcomes.

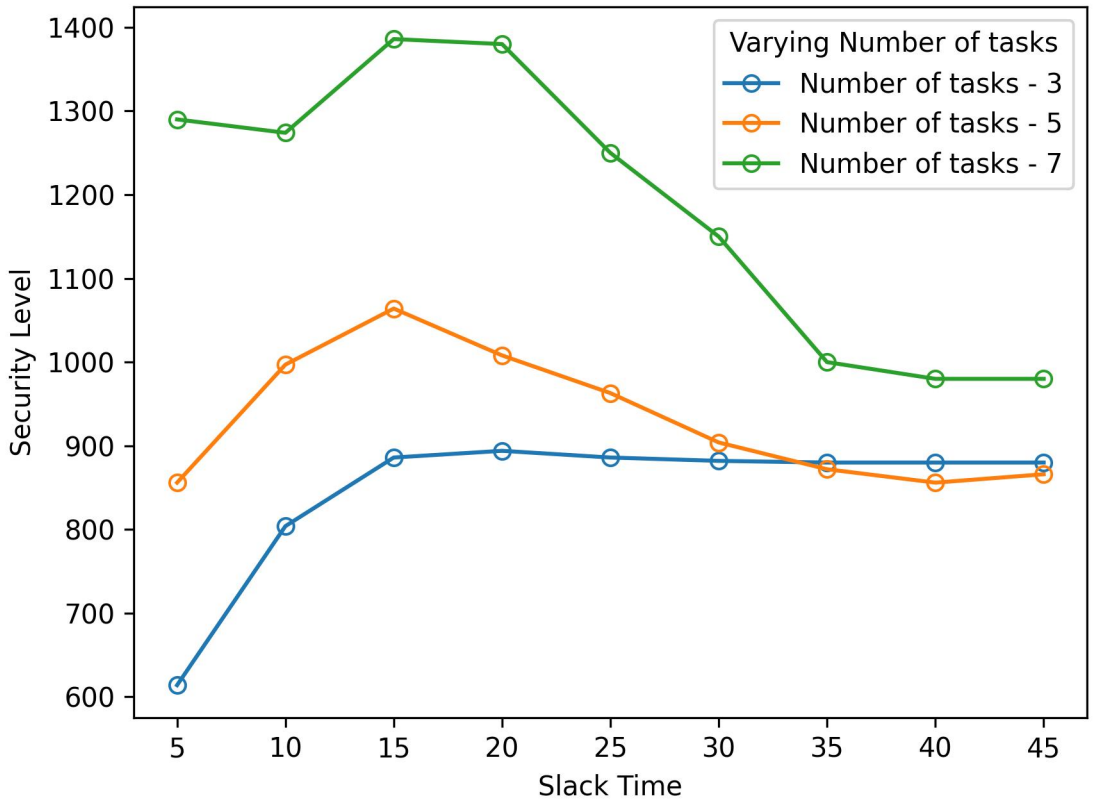

Figure 5.5: Varying Number of Tasks per TaskSet with FPTAS

5.2 Summary of Results

We presented a comparative analysis of three algorithms for providing security to real-time embedded systems. The algorithms considered are the optimal dynamic programming approach, the FPTAS approximation scheme, and a greedy-heuristic approach. The experiments were performed using synthetic task sets and security mechanisms that were generated using the UUnifast-Discard Algorithm. The results of the experiments were evaluated based on the maximum possible security level that can be achieved in the system, subject to the availability of system slack. The analysis shows that the dynamic programming approach produces the best results but is time-consuming and memory-intensive. The FPTAS approximation scheme produces near-optimal results and strikes a balance between the security levels assigned to different tasks in the system. The greedy-heuristic approach produces results that are somewhere between those of the optimal and FPTAS approaches but does not guarantee that all tasks in the system receive security. The experiments demonstrate that the FPTAS approach is a good compromise between optimality and efficiency.

Chapter 6

Conclusions

In this paper, we presented a real-time/security task model for CPS such that it is both timely and secure from the ground up. The proposed task model supports frame-based periodic task sets and was studied in detail with the help of two algorithms. First, we developed a knapsack-based approach. We implemented a full-polynomial time approximation scheme that finds a near-optimal solution for the knapsack problem. We confirm that the algorithm has a small approximation error of (1-ϵ). Second, we also proposed a greedy-heuristic approach that picks security mechanisms greedily. We then test and validate the two algorithms and also compare them with the state-of-the-art dynamic programming solution. Though the DP solution provides an optimal solution to the knapsack problem, it is memory intensive and runs in pseudo-polynomial time. In the future, we plan on augmenting the proposed task model to accommodate not only frame-based task sets but also periodic and sporadic task sets.

Bibliography

[1] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 110–124, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15031-9.

[2] M Bansal and V Venkaiah. Improved fully polynomial time approximation scheme for the 0-1 multiple-choice knapsack problem. *International Institute of Information Technology Tech Report*, pages 1–9, 2004.

[3] Steffen Borgwardt. Knapsack Problem Algorithms - CU Denver Optimization Student Wiki. http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Knapsack_Problem_Algorithms, May 2017.

[4] Giorgio C Buttazzo. *Hard real-time computing systems*. Real-Time Systems Series. Springer, New York, NY, 3 edition, September 2011.

[5] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[6] Shane S. Clark and Kevin Fu. Recent results in computer security for medical devices. In Konstantina S. Nikita, James C. Lin, Dimitrios I. Fotiadis, and Maria-Teresa Arredondo Waldmeyer, editors, *Wireless Mobile Communication and Healthcare*, pages 111–118, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29734-2.

[7] Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real Time Syst.*, 47 (1):1–40, 2011. doi: 10.1007/s11241-010-9106-5. URL https://doi.org/10.1007/s11241-010-9106-5.

[8] M. E. Dyer. An o(n) algorithm for the multiple-choice knapsack linear program. *Math. Program.*, 29(1):57–63, may 1984. ISSN 0025-5610. doi: 10.1007/BF02591729. URL https://doi.org/10.1007/BF02591729.

[9] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, symantec corp., security response*, 5(6):29, 2011.

[10] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers Security*, 28(1):18–28, 2009. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2008.08.003. URL https://www.sciencedirect.com/science/article/pii/S0167404808000692.

[11] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg", 2004. ISBN 978-3-540-24777-7.

[12] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010. doi: 10.1109/SP.2010.34.

[13] Man Lin, Li Xu, Laurence T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. Static security optimization for real-time systems. *IEEE Transactions on Industrial Informatics*, 5(1):22–37, 2009. doi: 10.1109/TII.2009.2014055.

[14] D. Lo, M. Ismail, Tao Chen, and G. Suh. Slack-aware opportunistic monitoring for real-time systems. In *2014 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 203–214, Los Alamitos, CA, USA, apr 2014. IEEE Computer Society. doi: 10.1109/RTAS.2014.6926003. URL https://doi.ieeecomputersociety.org/10.1109/RTAS.2014.6926003.

[15] Yue Ma, Wei Jiang, Nan Sang, and Xia Zhang. Arcsm: A distributed feedback control mechanism for security-critical real-time system. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 379–386, 2012. doi: 10.1109/ISPA.2012.56.

[16] Yue Ma, Wei Jiang, Nan Sang, and Ziguo Zhong. An adaptive risk control and security management for embedded real-time system. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 11–17, 2012. doi: 10.1109/ARES.2012.45.

[17] Yue Ma, Nan Sang, Wei Jiang, and Lei Zhang. Feedback-controlled security-aware and energy-efficient scheduling for real-time embedded systems. In James J. (Jong Hyuk) Park, Young-Sik Jeong, Sang Oh Park, and Hsing-Chung Chen, editors, *Embedded and Multimedia Computing Technology and Service*, pages 255–268, Dordrecht, 2012. Springer Netherlands. ISBN 978-94-007-5076-0.

[18] Tanmaya Mishra, Thidapat Chantem, and Ryan Gerdes. Teecheck: Securing intra-vehicular communication using trusted execution. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, RTNS 2020, page 128–138, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375931. doi: 10.1145/3394810.3394822. URL https://doi.org/10.1145/3394810.3394822.

[19] Tanmaya Mishra, Thidapat Chantem, and Ryan Gerdes. Survey of control-flow integrity techniques for real-time embedded systems. *ACM Trans. Embed. Comput. Syst.*, 21(4),

oct 2022. ISSN 1539-9087. doi: 10.1145/3538275. URL https://doi.org/10.1145/3538275.

[20] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2012.05.003. URL https://www.sciencedirect.com/science/article/pii/S1084804512001178.

[21] Sibin Mohan, Man-Ki Yoon, Rodolfo Pellizzoni, and Rakesh B. Bobba. Integrating security constraints into fixed priority real-time schedulers. *Real-Time Systems*, 52:644 – 674, 2016.

[22] Rodolfo Pellizzoni, Neda Paryab, Man-Ki Yoon, Stanley Bak, Sibin Mohan, and Rakesh B. Bobba. A generalized model for preventing information leakage in hard real-time systems. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 271–282, 2015. doi: 10.1109/RTAS.2015.7108450.

[23] Colin Percival. Cache missing for fun and profit. 2005.

[24] Tao Xie and Xiao Qin. Improving security for periodic tasks in embedded systems through scheduling. *ACM Trans. Embed. Comput. Syst.*, 6(3):20–es, jul 2007. ISSN 1539-9087. doi: 10.1145/1275986.1275992. URL https://doi.org/10.1145/1275986.1275992.

[25] Man-Ki Yoon, S. Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. Securecore: A multicore-based intrusion detection architecture for real-time embedded systems. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 21–32, Los Alamitos, CA, USA, apr 2013. IEEE Computer Society.

doi: 10.1109/RTAS.2013.6531076. URL https://doi.ieeecomputersociety.org/10.1109/RTAS.2013.6531076.

[26] Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016. doi: 10.1109/RTAS.2016.7461362.

[27] Eitan Zemel. An o(n) algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters*, 18(3):123–128, 1984. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(84)90014-0. URL https://www.sciencedirect.com/science/article/pii/0020019084900140.

[28] Xia Zhang, Jinyu Zhan, Wei Jiang, and Yue Ma. A vulnerability optimization method for security-critical real-time systems. In *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*, pages 215–221, 2013. doi: 10.1109/NAS.2013.34.