On Scheduler Side-Channels in Dynamic-Priority Real-Time Systems

Chien-Ying Chen*, Sibin Mohan*, Rodolfo Pellizzoni† and Rakesh B. Bobba‡

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
†Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada
‡School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Email: \{*cchen140, *sibin\}@illinois.edu, †rodolfo.pellizzoni@uwaterloo.ca, ‡rakesh.bobba@oregonstate.edu

Abstract—While the existence of scheduler side-channels has been demonstrated recently for fixed-priority real-time systems (RTS), there have been no similar explorations for dynamic-priority systems. The dynamic nature of such scheduling algorithms, e.g., EDF, poses a significant challenge in this regard. In this paper we demonstrate that side-channels exist in dynamic priority real-time systems. Using this side-channel, our proposed DyPS algorithm is able to effectively infer, with high precision, critical task information from the vantage point of an unprivileged (user space) task. Apart from demonstrating the effectiveness of DyPS, we also explore the various factors that impact such attack algorithms using a large number of synthetic task sets. We also compare against the state-of-the-art and demonstrate that our proposed DyPS algorithms outperform the ScheduLeak algorithms in attacking the EDF RTS.

I. INTRODUCTION

Due to the increased deployment of safety-critical systems with timing criticality (e.g., autonomous cars, delivery drones, industrial robots, implantable medical devices, power grid components), security for such systems becomes crucial. Until recently, security has been an afterthought in the design of real-time systems (RTS). However, the ever-increasing demand for using commodity-off-the-shelf (COTS) components and the demonstration of a variety of attacks against such systems in the field [1], [2], [3], [4], [5], [6], [7], [8] necessitates the need for a better understanding and classification of security threats aimed at RTS.

Side-channels that leak critical information about task behavior via system schedules in RTS has recently gained attention, including methods to protect against them [9], [10], [11], [12]. In particular, the ScheduLeak algorithms [12] demonstrate that scheduler side-channels can be exploited by an unprivileged task ("observer task") to leak important information such as task arrival times in fixed-priority RTS (FP RTS). This information was then used to predict future arrival times of critical tasks ("victim tasks"). While the leakage of such information seems to be subtle, knowledge about future arrival times, especially for critical tasks, can help increase the effectiveness of other attacks by filtering out noisy data and extracting valuable information about the victim system. In fact, these types of attacks fall into the broader category of reconnaissance attacks. It has been shown that they can be the stepping stone for more sophisticated attacks [14], [15]. For instance, the ScheduLeak paper demonstrated how cache timing attacks or even the ability to take control of autonomous drones become much simpler once the victim task’s future behavior is made available to adversaries. One main drawback of ScheduLeak is that it has only been demonstrated for FP RTS. This significantly limits the types of systems where such attacks can be launched. Directly applying ScheduLeak to dynamic-priority RTS also does not work well — as evidenced by the precision of inference (for EDF; the grey bars) in Figure 1. Hence, we need to develop algorithms that are targeted towards dynamic-priority real-time systems.

Figure 1: The results of employing the state-of-the-art (ScheduLeak) and the proposed DyPS algorithms in the scheduler side-channel attack against the EDF RTS. The end (mean) inference precision of ScheduLeak is 0.54 that is only slightly better than a naive attack with random guesses.

One important challenge to leaking information via scheduler side-channels in dynamic priority RTS, e.g., the earliest-deadline first (EDF) algorithm, is that (relative) task priorities are not constant and vary at run-time. In the ScheduLeak attack [12] that targets FP RTS, the observer task has a priority lower than the victim task at all times. This determinism ensures that the execution of the observer task is always preempted or delayed by the victim task when both are ready to run and this is vital for inferring the arrival times of the victim. In contrast, in an EDF RTS, task priorities are determined dynamically based on each job’s absolute-relative deadline at each scheduling point. That is, no task will always have a higher priority relative to another task in the system. Hence, the ScheduLeak assumption about a persistent (rela-
tive) priority relationship between any two tasks in the system becomes invalid in EDF RTS. Consequently, the ScheduLeak algorithms designed for FP RTS will fail (or have much lower success rates) while dealing with EDF RTS.

We present the Dynamic-Priority ScheduLeak (DyPS) algorithms that (a) demonstrate the existence of scheduler side-channels in the Earliest Deadline First (EDF) scheduling algorithm and (b) make use of such information to extract critical task information. DyPS builds on top of ScheduLeak but introduces additional steps and analyses to overcome the challenges w.r.t. dynamic priority schedulers. We then present our findings on how to craft effective attacks — that builds upon the pairwise relationships between tasks in the system. In addition, we also explore conditions that could limit the scope of this attack. Finally, our evaluation demonstrates the effectiveness of DyPS and also compares it to the state-of-the-art (ScheduLeak) — we are able to match the performance of ScheduLeak with a difference in precision of less than 0.1%.

To summarize, this paper makes the following contributions:
1) DyPS — a set of attack algorithms that overcome the uncertainty in the EDF scheduling to accurately extract critical task information from the scheduler side-channels [Section III];
2) Analyses and metrics to understand the factors that can influence an attacker’s ability to carry out a successful DyPS attack [Section IV];
3) Comparison with the state-of-the-art scheduler side-channel attack algorithms [Section V].

II. SYSTEM AND ADVERSARY MODEL

A. System Model

In this paper, a discrete time model [16] is considered. We assume that there exists a system timer (e.g., a CLOCK_MONOTONIC-based timer in Linux, a global tick counter in FreeRTOS) that produces time ticks that the EDF scheduler can use and that the tick count is an integer. We further assume that a unit of time is equal to a tick time and all system parameters are multiples of this tick time. We denote an interval starting from time point \( a \) and ending at time point \( b \) by \([a, b)\) or \([a, b-1]\) (and hence its length equals \( b-a \)).

We consider a uni-processor, single-core, preemptive, dynamic-priority RTS running the EDF scheduling algorithm. The system consists of \( n \) real-time tasks \( \Gamma = \{\tau_1, \tau_2, ..., \tau_n\} \), each of which can be either a periodic or a sporadic task. A task \( \tau_i \) is modeled by \( C_i, T_i, D_i, \phi_i \) where \( C_i \) is the worst-case execution time (WCET), \( T_i \) is the period (or the minimum inter-arrival time for a sporadic task), \( D_i \) is the relative deadline and \( \phi_i \) is the task phase.\(^1\) We assume that every task has a distinct period (or the minimum inter-arrival time) and that \( D_i = T_i \) [17]. We denote the \( k \)-th job of the task \( \tau_i \) by \( \tau_i^k \) and it is modeled by \( c_i^k, a_i^k, s_i^k, d_i^k \) where \( c_i^k \) denotes the execution time (\( c_i^k \leq C_i \)), \( a_i^k \) is the absolute arrival time, \( s_i^k \) is the start time and \( d_i^k \) is the absolute deadline. For simplicity, we use \( c_i, a_i, s_i, d_i \) when referring to an arbitrary job of \( \tau_i \) if the job ordering is unimportant. Furthermore, we use “task” and “job” interchangeably. In this paper, we only consider the task set that is schedulable by the EDF scheduling algorithm. Therefore, \( \sum_{\tau_i \in \Gamma} \frac{C_i}{T_i} \leq 1 \). We assume that the task release jitter is negligible, and thus \( a_i^{k+1} - a_i^k = T_i \) for a periodic task and \( a_i^{k+1} - a_i^k \geq T_i \) for a sporadic task.

In the EDF scheduling algorithm, a job with smaller absolute deadline gets to run first and is considered to have higher priority among other ready jobs that have greater absolute deadlines. In the case that multiple jobs in the ready queue have the same absolute deadline, they are considered to have the same priority and the EDF scheduler randomly selects one of the jobs to run. We define a task’s “execution interval” to be an interval during which the task runs continuously.

B. Adversary Model

Similar to the adversary model introduced in existing work on scheduler side-channels in RTS [12, 9, 13], we assume that the attacker is interested in learning the task phase (and inferring the future arrival time points) of a critical, periodic task (the victim task, denoted by \( \tau_v \)) in the system. This is considered to be part of a reconnaissance phase that can be a part of a larger attack. Such an attack will benefit from the inferences of the task’s future arrival time points. The attacker launches the proposed DyPS attack algorithms that exploit the scheduler side-channels using an unprivileged, periodic task (the observer task, denoted by \( \tau_o \)) running on the same victim system. Once the inference of the victim task’s phase is obtained, it is up to the attacker to decide if further attacks should be launched using the same observer task or via other attack surfaces. The ultimate goal of the attacks varies with the adversaries. It has been shown that the inferred future arrival time points can be used to help increase the success rate of an attack (e.g., overriding the control of a rover system) or filtering out noisy data while extracting valuable information (e.g., monitoring the task’s execution behavior by using a cache-timing side-channel attack) [12].

The DyPS attack requires only the observer task to ensure the success for the attack algorithms introduced in this paper. We assume that the observer task has access to a system timer that has a resolution that is coarser than or equal to a time tick. The observer task uses the timestamps read from such a system timer to reconstruct its own execution intervals and infer the victim task’s phase. However, this method only works when the victim task has a priority higher than the observer task [12 Theorem 1], which is not always true under the EDF scheduling algorithm. More specifically, in the case of EDF RTS, we are more interested in the priority relationships between tasks at the instant when the victim task arrives. To
better clarify the relation between the observer task and the victim task, we define the term “observability” as follows:

**Definition 1.** (Observability) A victim task’s arrival at $a_v$ is said to be observable by the observer task if the observer task has a priority lower than the victim task at $a_v$.

In an FP RTS, it is trivial to see that every arrival of the victim task is observable by the observer task if the victim task has a fixed, higher priority than the observer task. In an EDF RTS, it depends on both the tasks’ periods and the absolute deadlines at run-time. We first determine the observability of a given observer and victim task pair by using the following theorem:

**Theorem 1.** For the EDF scheduling algorithm, given an observer task $τ_o$ and a victim task $τ_v$ whose periods are $T_o$ and $T_v$, respectively and $T_o \neq T_v$, the victim task’s arrivals may be observable by the observer task only if $T_o > T_v$.

**Proof.** Definition 1 for EDF means that the observer task has a larger deadline when the victim task arrives. That is, $a_o \leq a_v$ and $d_o > d_v$ (or $a_o + T_o > a_v + T_v$) for some jobs of $τ_o$ and $τ_v$. Rewriting the above deadline inequality as $T_o - T_v > a_v - a_o$ indicates that $T_o$ must be greater than $T_v$ since $a_v - a_o \geq 0$. Now let’s consider the case when $T_o < T_v$. It can be seen that when both tasks arrive at the same time (which is the case when the observer task has the largest possible deadline relative to the victim task), the observer task still has a deadline smaller than the victim task ($i.e.$, $a_o = a_v$ and thus $a_o + T_o < a_v + T_v \Rightarrow d_o < d_v$). Therefore, no victim task’s arrival can be observed by the observer task if $T_o < T_v$.

Based on Theorem 1 we make an assumption that the observer task must have a period larger than the victim task ($i.e.$, $T_o > T_v$) in this paper.

**III. THE SCHEDULER SIDE-CHANNELS IN EDF**

The scheduler side-channels in the FP RTS enable an unprivileged, low-priority task to learn precise timing information of a periodic, critical task. Similar scheduler side-channels exist in the EDF RTS due to the fact that both types of RTS are preemption-based systems. However, because of the dynamic nature, there are additional conditions and restrictions to be considered for making the attack succeed under the EDF scheduling algorithm. In this section, we present these constraints along with the details of the proposed DyPS algorithms.

**A. Challenges and Overview**

In this paper, the attacker’s goal is to infer the victim task’s phase and then predict its future arrival time points. This is achieved by allowing the observer task to reconstruct and analyze its own execution intervals. When the victim task arrives with a priority higher than the job of the observer task that has been scheduled, the execution of the latter is either delayed or preempted. As a result, the victim task’s execution (including the arrival instant) is enclosed in the observer task’s execution intervals. By reconstructing execution intervals for a sufficiently long duration (see Section V-C2 for the evaluation of the attack duration), it is possible to infer the victim task’s phase. Yet, even if we assume that $T_o > T_v$ (Theorem 1) for a given observer and victim task pair, it is not guaranteed that every arrival of the victim task is observable by the observer task due to the dynamic priority in EDF.

**Example 1.** Consider an observer task $τ_o$ ($T_o = 10, C_o = 4$) and a victim task $τ_v$ ($T_v = 8, C_v = 2$). Without making any assumption on the task phases, Figure 2 demonstrates how dynamic priorities impact the observability in EDF schedules. In Figure 2(a), both $τ_o$ and $τ_v$ arrive at the same time point ($i.e.$, $a_o = a_v$). Since $τ_o$ has a higher priority (because $d_v < d_o$), the execution of $τ_v$ delays $τ_o$ that is supposed to start at $a_o$. As a result, the arrival at $a_v$ can be observed by $τ_o$. In Figure 2(b), $τ_o$ arrives at a point when $a_o < d_v$. As $τ_o$ is currently executing and has a higher priority at the arrival point $a_v$, the execution of $τ_v$ is delayed by the execution of $τ_o$. Consequently, the observer task fails to observe the victim task’s arrival at $a_v$.

Apart from the fact that not all the observer task’s execution intervals encapsulate the victim task’s arrivals, the above example also hints at the fact that a part of an execution interval may still be admissible even when such an execution interval is considered invalid w.r.t. the observation of the victim task’s arrivals. To illustrate, let’s consider the observer task’s execution interval starting at $a_o$ in Figure 2(b). It is in fact safe to leverage the first half of the execution interval to infer there is no arrival. This is because if the victim task arrived within this interval it would have higher priority (earlier deadline) and would preempt the observer task. But considering the second half of the execution interval to make the same inference would cause a false negative observation ($i.e.$, no arrival is inferred while there is one) and mislead the attack results. To identify the part of the execution interval that is valid for observing the victim task’s arrivals, we present the following theorem:

**Theorem 2.** For a given job of the observer task arriving at
Reconstruct the observer task’s phase \( \phi_0 \)

Reconstruct the valid part of the execution intervals of \( \tau_0 \)

Compute candidate time slots

Infer the victim task’s phase \( \phi_v \)

Predict future arrival time points

Figure 3: An overview of the attack steps in the DyPS algorithms. The first two steps are specific to dealing with the dynamic nature in EDF. Step 2 and 3 are identical to ScheduLeak.

\[ \phi \]

\[ T \]

\[ d \]

\[ a \]

\[ T_0 \]

\[ s \]

\[ k \]

\[ C \]

\[ \tilde{\phi} \]

\[ \phi(o) \]

\[ \phi(v) \]

\[ \phi_0 \]

\[ \phi_v \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]

\[ \tau_0 \]
where $T_o$ and $p$ are known.

In the above proposition, the given start time pair, $s^k_o$ and $s^{k+p}_o$, represents two jobs differing in $p$ periods. Therefore, the start time of the $(k+p)$-th job can be shifted to the same period as the $k$-th job by $(s^{k+p}_o - p \cdot T_o)$. As a result, the two start times in the same period (i.e., $s^k_o$ and the shifted $s^{k+p}_o - p \cdot T_o$) are comparable. The smaller one is closer to its arrival time since $a_o \leq s_o$.

By employing Proposition[1] we can determine the start time that is closest to its arrival time using $S_{o \text{observed}}$. The inference of the task phase for the observer task is computed by

$$\phi_o = \min(s^{k+p}_o - p \cdot T_o \mid 0 \leq p < m) \mod T_o \quad (3)$$

where $T_o$ is the period known to the attacker and $k$ can be unknown. Then, given a start time $s_o$, the attacker can compute its projected arrival time $\tilde{a}_o$ by

$$\tilde{a}_o = s_o - \tilde{J}_o \quad (4)$$

where $\tilde{J}_o = (s_o - \phi_o) \mod T_o$ represents the delay such a job may have experienced.

**Example 2.** Consider the observer task $\tau_o$ in a task set of 4 periodic tasks (extended from Example[1]) as shown in the table below.

| $T_i$ | $C_i$ | $\phi_i$ |
|-------|-------|----------|
| $\tau_1$ | 15 | 2 |
| $\tau_2$ | 10 | 4 |
| $\tau_3$ | 8  | 2 |
| $\tau_4$ | 6  | 1 |

Let’s assume the system begins at $t = 0$ and the observer task starts collecting its start times for 10 instances from $t = 41$. The collected start times are \{41, 53, 61, 71, 81, 92, 101, 111, 121, 133\}. By using Equation[3] the observer task’s phase $\phi_o$ can be computed by

$$\phi_o = \min(41, 43, 41, 41, 42, 41, 41, 41, 43) \mod 10 = 1.$$  

In this example, $\phi_o = \phi_o = 1$ and thus the correct observer task’s phase is obtained.

**C. Reconstructing The Observer Task’s Execution Intervals**

Based on Theorem[2] and Equation[4] we developed the following proposition to reconstruct the execution intervals in a period:

**Proposition 2.** Assume that, in a DyPS attack, the attacker has reconstructed the observer task’s phase as $\phi_o$. Then, for a given job of the observer task starting at $s_o$, the observer task reconstructs only the execution intervals within $[\tilde{a}_o, \tilde{a}_o + T_o - T_v]$ where $\tilde{a}_o$ is calculated using Equation[4].

To implement Proposition[2] we employ a known reconstruction algorithm[2] Algorithm 1 but make the following modifications: (i) at the beginning of the execution of each period, we let the observer task compute $\tilde{a}_o$ by using Equation[4] (ii) the observer task stops reconstructing execution intervals in a period if the current time stamp exceeds $\tilde{a}_o + T_o - T_v$.

The end result of this step is a set of reconstructed execution intervals, denoted by $E^{recon}_{o} = \{e^1_o, e^2_o, e^3_o, \ldots\}$ where $e^r_o := [\text{begin}^r_o, \text{end}^r_o]$ is the $r$-th reconstructed execution interval starting at the time point $\text{begin}^r_o$ and ending at the time point $\text{end}^r_o$. Note that the number of reconstructed execution intervals are dependent on the duration of the attack which is determined by the attacker. The impact of the attack duration is evaluated in Section V.C2

**D. Computing The Candidates**

To compute the candidate time points for the phase of the victim task, the reconstructed execution intervals are organized on a timeline with length equal to the victim task’s period $T_v$. To facilitate understanding, let us use the schedule ladder diagram[12] of width $T_v$ to illustrate how the reconstructed execution intervals are processed. On a schedule ladder diagram, the victim task’s arrivals are always present in the same column (since the width equals $T_v$). Let’s define such a column as the “true arrival column” that has an offset of $\tau_o$ from the leftmost time column. As the reconstructed execution intervals of $\tau_o$ are ensured to have priorities lower than the victim task, those execution intervals will not appear in the true arrival column (because otherwise the victim task would have executed instead). In other words, the time columns where the reconstructed execution intervals of $\tau_o$ appear even once cannot be the true arrival column.

Given a reconstructed execution interval $e^r_o \in E^{recon}_o$, the time columns where $e^r_o$ is present are determined by \{t mod $T_v \mid \text{begin}^r_o \leq t < \text{end}^r_o \land t \in \mathbb{Z}\}. For simplicity, let’s define $e^r_o \mod T_v := \{t \mod T_v \mid \text{begin}^r_o \leq t < \text{end}^r_o \land t \in \mathbb{Z}\}. Therefore, the time columns where the reconstructed execution intervals appear at least once can be calculated by \bigcup_{e^r_o \in E^{recon}_o} (e^r_o \mod T_v) which represents a set of “false” time columns that do not include the true arrival time column. Thus, the set of candidate time columns can be obtained by

$$\{\text{col} \mid 0 \leq \text{col} < T_v \land \text{col} \in \mathbb{Z}\} - \bigcup_{e^r_o \in E^{recon}_o} (e^r_o \mod T_v) \quad (5)$$

**E. Inferring The Victim Task’s Phase**

Next, we take the beginning of the longest contiguous time columns in the candidate list as the inference of the victim task’s phase, $\tilde{\phi}_v$. Then, the future arrival time of the victim task can be calculated by $\phi_v + k \cdot T_v$ where $k$ is the desired arrival number. Alternatively, given a time point $t$, the subsequent arrival time of the victim task can be predicted by

$$t + (\tilde{\phi}_v - t) \mod T_v \quad (6)$$

**Example 3.** Let’s consider the task set from Example[2]. Assume that the observer task has collected its execution intervals for a duration of $\text{LCM}(T_o, T_v)$ (i.e., 40 time units in this example) since the job starts at $t = 41$. The reconstructed execution intervals are $E^{recon}_o = \{41, 42, 61, 63, 71, 73\}$ and they correspond to the time columns \{1\}, \{5, 6\} and

\footnote{As we will see in Section V.C2 the attack duration is evaluated with using $\text{LCM}(T_o, T_v)$ as an unit since the offset between the arrivals of $\tau_o$ and $\tau_v$ resets every $\text{LCM}(T_o, T_v)$}
Figure 4: A schedule ladder diagram demonstrated for Example 3. Both valid and invalid execution intervals are plotted in the diagram for reference. In the DyPS algorithms, only the valid execution intervals are taken into account for inferring the victim task’s phase (Proposition 2).

Figure 5: An example of the arrival time correlation between \( \tau_0 \) and \( \tau_1 \) for Theorem 3. In this example, an observer task \( \tau_0 \) \((T_o = 10)\) and a periodic task \( \tau_1 \) \((T_1 = 8)\) are considered. It shows that the projected arrivals of \( \tau_0 \) appear every \( \sigma(T_o, T_1) T_1 = 2 \) time columns.

A. Impact on The Reconstruction of Observer Task’s Phase

In Section III-B we presented algorithms that can reconstruct the observer task’s phase by using the start times of its own jobs. When there exists at least one job of the observer task whose arrival time equals its start time (i.e., the job is not delayed by any higher priority task and thus \( a_o = s_o \)), the correct task phase can be reconstructed. On the other hand, the correct inference cannot be made if the start of the observer task jobs consistently experience delays in every period. Here we explore and characterize the factors that may contribute to the delays in the observer task’s start times by analyzing a simple task set with just two tasks.

**Theorem 3.** Given an observer task \( \tau_o \) and a periodic task \( \tau_i \), the maximum proportion of the arrivals of \( \tau_o \) that may experience interference solely due to task \( \tau_i \) is computed by

\[
\psi(\tau_o, \tau_i) = \left[ \frac{u_i}{\sigma(\tau_o, \tau_i)} \right] \sigma(\tau_o, \tau_i)
\]

where \( u_i = \frac{C_i}{T_i} \) is the utilization of \( \tau_i \), \( \sigma(\tau_o, \tau_i) = \frac{T_o}{LCM(T_o, T_i)} \) is the inverse of the number of the arrivals of \( \tau_o \) in a \( LCM(T_o, T_i) \) and the resulting \( \psi(\tau_o, \tau_i) \) is a fraction in the range \( 0 < \psi(\tau_o, \tau_i) \leq 1 \).

**Proof.** Since both tasks are periodic, the schedule of the two tasks repeats after their least common multiple, \( LCM(T_o, T_i) \). There are \( LCM(T_o, T_i) \) arrivals of \( \tau_o \) in a \( LCM(T_o, T_i) \). By projecting these arrivals of \( \tau_o \) onto a timeline with length equal to the period of \( \tau_i \) an arrival of \( \tau_o \) appears every \( \frac{1}{LCM(T_o, T_i)} T_i = \sigma(\tau_o, \tau_i) T_i \) time units and repeats after each \( LCM(T_o, T_i) \) (Observation 2), as illustrated in Figure 5. The most number of arrivals of \( \tau_o \) that may experience interference due to the execution of \( \tau_i \) in one \( LCM(T_o, T_i) \) is computed by

\[
\psi(\tau_o, \tau_i) = \left[ \frac{C_i}{\sigma(\tau_o, \tau_i) T_i} \right] \sigma(\tau_o, \tau_i)
\]

which happens when there exists an instant at which both tasks arrive at the same time, e.g., when \( \phi_o = \phi_i \) or \( (\phi_o \mod T_i) = \phi_i \). In this case, the maximum proportion of the arrivals of \( \tau_o \) may be interfered by \( \tau_i \) in one \( LCM(T_o, T_i) \) can be computed by

\[
\psi(\tau_o, \tau_i) = \left[ \frac{u_i}{\sigma(\tau_o, \tau_i)} \right] \sigma(\tau_o, \tau_i).
\]

Since this relation is the same across all \( LCM(T_o, T_i) \), the calculated \( \psi(\tau_o, \tau_i) \) applies to the whole schedule.

When \( \psi(\tau_o, \tau_i) < 1 \), it means that at least \( 1 - \psi(\tau_o, \tau_i) \) of the arrivals of \( \tau_o \) are not interfered by \( \tau_i \). In contrast, when \( \psi(\tau_o, \tau_i) = 1 \), it is possible for the execution of \( \tau_i \) to interfere all the arrivals of \( \tau_o \), which would result in an inaccurate \( \phi_o \). Intuitively, it can happen when \( \tau_i \) has a period in harmony with that of \( \tau_o \). An example is given below.
will not experience any interference. In such a case, phases as well as the run-time variations. Therefore, the actual impact is highly dependent on the task and reconstructing the correct task phase becomes possible.

If \( \phi_1 = 0 \) and \( \phi_o = 1 \), all the arrivals of \( \tau_o \) experience interference due to the harmonic task \( \tau_1 \).

If \( \phi_1 = 0 \) and \( \phi_o = 2 \), no arrival of \( \tau_o \) experiences interference due to the harmonic task \( \tau_1 \).

Example 4. Consider an observer task \( \tau_o \) (\( T_o = 10, C_o = 4 \)) and a task \( \tau_1 \) (\( T_1 = 5, C_1 = 2 \)). The two tasks are in harmony because \( T_o \mod T_1 = 0 \). Then \( \psi(\tau_o, \tau_1) \) is computed by

\[
\psi(\tau_o, \tau_1) = \left\lceil \frac{u_1}{\sigma(\tau_o, \tau_1)} \right\rceil = \left\lceil \frac{u_1}{\sigma(\tau_o, \tau_1)} \right\rceil = 1
\]

which indicates that all the arrivals of \( \tau_o \) may experience interference due to \( \tau_1 \). Figure 6(a) illustrates two possible schedules for the given task set. Figure 6(a) shows the case where \( \phi_1 = 0 \) and \( \phi_o = 0 \), all the arrivals of \( \tau_o \) are interfered by \( \tau_1 \). Figure 6(b) shows the case where \( \phi_o = 2 \) and \( \phi_1 = 0 \), all the arrivals of \( \tau_o \) are no longer interfered by \( \tau_1 \).

This example shows a crucial fact that \( \psi(\tau_o, \tau_1) \) only represents the upper bound of the interference (when only one task is considered). That is, having \( \psi(\tau_o, \tau_i) = 1 \) does not mean all the arrivals of \( \tau_o \) are absolutely interfered. With the same task set but different task phases (which can vary across systems and every time the system restarts), the impact of the interference can be quite different, as shown in Example 6.

On the other hand, the schedules presented in Figure 6 are generated based on WCETs. However, the actual execution times at run-time in real systems can vary throughout. As a result, the run-time task utilization is in fact smaller and, based on Equation 7, the proportion of arrivals of the observer task is also smaller. Taking the schedule in Figure 6(a) as an example, if the first instance’s execution time of \( \tau_1 \) is \( c_1^o = 1 \) (rather than its WCET), then the arrival \( a_o^1 \) of the observer task will not experience any interference. In such a case, \( s_o^1 = a_o^1 \) and reconstructing the correct task phase becomes possible. Therefore, the actual impact is highly dependent on the task phases as well as the run-time variations.

B. Coverage Ratio in EDF

The coverage ratio \( \mathcal{C}(\tau_o, \tau_o) = \frac{c_o^\tau}{\text{GCD}(T_o, T_o)} \) (Definition 1) is used to estimate the proportion of the time columns that can be covered by the execution of the observer task in the FP RTS. When a given observer task and victim task pair satisfies \( \mathcal{C}(\tau_o, \tau_o) \geq 1 \), the execution of the observer task may cover all the time columns on the schedule ladder diagram and observing the victim task’s arrivals is possible. Since the coverage ratio and the inference precision have a positive correlation, it is useful for evaluating the attacker’s capabilities against the victim task. While the idea of the coverage ratio can be applied in the case of DyPS, the calculation must be revised to reflect the scope of the reconstructed execution intervals specified in Proposition 2 before it can be used. Therefore, we redefine the the coverage ratio for DyPS as follows:

**Definition 2.** \( \mathcal{C}_{DyPS}(\tau_o, \tau_o) \) (DyPS Coverage Ratio) The coverage ratio of DyPS in EDF, denoted by \( \mathcal{C}_{DyPS}(\tau_o, \tau_o) \), is computed by

\[
\mathcal{C}_{DyPS}(\tau_o, \tau_o) = \frac{\min(C_o, T_o - T_v)}{\text{GCD}(T_o, T_v)} \tag{8}
\]

It represents the proportion of the time columns where the observer task’s (valid) execution can potentially be present in the schedule ladder diagram. If all \( T_o \) time columns can be covered by the observer task, then \( \mathcal{C}_{DyPS}(\tau_o, \tau_o) \geq 1 \). Otherwise \( 0 < \mathcal{C}_{DyPS}(\tau_o, \tau_o) < 1 \).

V. Evaluation

A. Evaluation Metrics

There are mainly two attack stages in the DyPS algorithms: (i) reconstructing \( \phi_o \) for determining valid execution intervals and (ii) inferring \( \phi_o \) for predicting future arrival time points. While both stages target the computing of a task’s phase, they have very different characteristics due to how they are inferred. We use two different metrics to evaluate the results from the two stages as defined next.

1) Reconstructing The Observer Task Phase \( \phi_o \): As introduced in Section III-B, the observer task’s phase is reconstructed based on the collected start times where \( s_o \geq a_o \) (i.e., the start times are always on the right of the corresponding arrival times), thus the distance between \( \phi_o \) and \( \phi_o \), denoted by \( \Delta \phi_o = (\phi_o - \phi_o) \mod T_o \), should always be positive. Based on this fact, we define the error ratio for \( \phi_o \) as follows:
The number of start times to infer correct $\phi_o$, denoted by $E^o$, is computed by

$$E^o = \frac{\Delta \tilde{\phi}_o}{T_o}$$

(9)

where $\Delta \tilde{\phi}_o = (\tilde{\phi}_o - \phi_o) \mod T_o$ represents the distance between $\phi_o$ and a projected $\tilde{\phi}_o$ on its right. The resulting $E^o$ value is a real number in the range $0 \leq E^o \leq 1$. A smaller $E^o$ means that the reconstructed $\phi_o$ has less error when compared to the true $\phi_o$.

Note that $E^o$ is bounded by 1 because the start times used for computing $\phi_o$ are bounded by $a_o < s_o < a_o + T_o$.

2) Inferring The Victim Task Phase $\phi_v$: The second stage of the attack in DyPS is to infer the task phase of the victim task. In contrast to the observer task’s phase, we are only concerned with how close the inference $\tilde{\phi}_v$ is to the actual $\phi_v$ and $\tilde{\phi}_v$ can be on either side of $\phi_v$. Inference precision $I^v$ of $\phi_v$ was introduced to evaluate the effectiveness of scheduler side-channel attacks. Here, we use the same metric but with a clearer equation for evaluation. The metric is defined as follows.

Definition 3. ($E^o$ Error Ratio of $\tilde{\phi}_o$) The error ratio of $\tilde{\phi}_o$, denoted by $E^o$, is computed by

$$E^o = \frac{\Delta \tilde{\phi}_o}{T_o}$$

(9)

where $\Delta \tilde{\phi}_o = (\tilde{\phi}_o - \phi_o) \mod T_o$ represents the distance between $\phi_o$ and a projected $\tilde{\phi}_o$ on its right. The resulting $E^o$ value is a real number in the range $0 \leq E^o \leq 1$. A smaller $E^o$ means that the reconstructed $\phi_o$ has less error when compared to the true $\phi_o$.

Note that $E^o$ is bounded by 1 because the start times used for computing $\phi_o$ are bounded by $a_o < s_o < a_o + T_o$.

Definition 4. ($I^v$ Inference Precision of $\tilde{\phi}_v$) The inference precision, denoted by $I^v$, is computed by

$$I^v = \frac{\varepsilon}{\frac{T_v}{2}} - 1$$

(10)

where $\varepsilon = |\tilde{\phi}_v - \phi_v|$. The resulting $I^v$ value is a real number in the range $0 \leq I^v \leq 1$. A larger $I^v$ indicates that the inference $\tilde{\phi}_v$ is more precise in inferring $\phi_v$.

B. Evaluation Setup

The DyPS algorithms are tested using synthetic task sets that are grouped by utilization from $\{0.001 + 0.1 \cdot x, 0.1 + 0.1 \cdot x\} | 0 \leq x < 9 \land x \in \mathbb{Z}$}. Each group contains 6 subgroups that have a fixed number of tasks from $\{5, 7, 9, 11, 13, 15\}$. A total of 100 task sets are generated for each subgroup. The utilization of each individual task in a task set is generated from a uniform distribution by using the UUniFast algorithm [18]. For each task in a task set, the period $T_i$ is randomly drawn from $[100, 1000]$ and the worst-case execution time $C_i$ is computed based on the generated task utilization and period. The task phase is randomly selected from $[0, T_i]$.

The observer task and the victim task in a task set are selected from the generated tasks based on the task periods. To illustrate, let us consider a task set consisting of $n$ tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ whose task IDs are ordered by their periods (i.e., $T_1 > T_2 > ... > T_n$). The observer task is then selected as the $(\lfloor n \cdot \frac{a_o}{T_o} \rfloor + 1)$-th task and the victim task is selected as the $(\lfloor n \cdot \frac{a_v}{T_v} \rfloor + 1)$-th task. This assignment ensures that $T_o > T_v$ (an assumption from Theorem 1) and that there exist other tasks with diverse periods (i.e., some with smaller periods and some with larger periods compared to $T_o$ and $T_v$). The task sets are generated with $C_{DyPS} \geq 1$ (except for the experiment that evaluates the impact of the DyPS coverage ratio). It is to examine the performance of the DyPS algorithms under the best case (i.e., all $T_i$ time columns on the schedule ladder diagram may be covered by the observer task’s execution.)

In each task set, 50% of the tasks are configured as sporadic tasks and the rest are periodic tasks. Thus, there are $\lfloor \frac{n}{2} \rfloor$ sporadic tasks in each task set. The generated task periods are used as the minimum inter-arrival times for the sporadic tasks. To vary the inter-arrival times for the sporadic tasks at run-time, we use a Poisson distribution in which the probability of each drawn occurrence (i.e., each inter-arrival time) is independent. During the simulation, a Poisson distribution with $\lambda = T_v \cdot 120\%$ as its mean value is used to generate the different inter-arrival times for a sporadic task $T_v$. To satisfy the given minimum inter-arrival time, we adjust the varied inter-arrival time to be $T_v$ if it becomes smaller than $T_v$.

To generate the different task execution times at run-time, a normal distribution is employed. For a task $T_v$, we fit a normal distribution $N(\mu = C_i \cdot 80\%, \sigma^2)$ with which the cumulative probability for $c_i \leq C_i$ is 99.99%. When a generated execution time exceeds its $C_i$, we adjust it to be $C_i$ to ensure the schedulability of the task set.

C. Simulation Results

1) Reconstructing The Observer Task’s Phase $\phi_o$: As introduced in Section III-B the observer task’s phase has to be reconstructed before inferring the victim task’s phase. Therefore, we evaluate the factors that impact the reconstruction of the observer task’s phase.

We first test the DyPS algorithms without any tasks that are in harmony with the observer task. The result shows that the correct $\phi_o$ (i.e., $\phi_o = \phi_o$ or $E^o = 0$) can be reconstructed in all of the tested 6000 task sets (without harmonic tasks). The number of start times that the DyPS algorithms process to get $E^o = 0$ is plotted in Figure 7. The figure shows a trend that it requires more start times for the DyPS algorithms to reconstruct correct $\phi_o$ when the utilization is higher. This
is because a higher utilization implies higher chances of preemptions and delays in task executions. Nevertheless, even in the worst case, of 6000 tested task sets, a reasonably small number of start times were required (i.e., 15 start times). This shows that the DyPS algorithms are practical for reconstructing $\phi_v$.

Next we evaluate the impact of harmonic tasks. As pointed out in Section IV-A, a task with a period that is in harmony with the observer task can contribute a constant delay to the observer task’s start times. Therefore, we regenerate 6000 task sets with at least one task in harmony with the observer task in each task set. The X-axis is the ratio of the estimated start time delays introduced by the harmonic tasks and the observer task’s period and the Y-axis is the error ratio. Each task set is tested with run-time variations and without run-time variations. The results show that the impact of the start time delays becomes subtle in the presence of run-time variations.

To understand the impact of the error ratio of $\hat{\phi}_v$ on the inference precision for $\hat{\phi}_v$, we test the aforementioned 6000 task sets with synthetically generated $\phi_v$ from $\{\phi_v + (0.01x)T_o \mid 0 \leq x \leq 10 \land x \in \mathbb{Z}\}$ which is expected to yield error ratio in $\{0, 0.01, ..., 0.1\}$. This range is chosen based on the experiment results shown in Figure 8 where overall error ratio is smaller than 0.034 with run-time variations. The experiment is carried out with a duration of $10 \cdot LCM(T_o,T_v)$ and with run-time variations enabled. Results shown in Figure 9 indicate that the error ratio has considerable negative impact on the inference precision. For example, inference precision $\phi_v^*$ reduces to 0.79 when $E^o = 0.01$ from $\phi_v^* = 0.98$ when $E^o = 0$. Nevertheless, considering only 0.7% of the task sets have $E^o > 0$ due to the run-time variations and the varied task phases, a high $E^o$ is arguably uncommon in real cases.

2) Inferring The Victim Task’s Phase $\phi_v$: We now focus on evaluating the inference precision of $\phi_v$. Note that the complete DyPS algorithms are tested in the experiments here. That is, the $\phi_v$ values are reconstructed as a part of the algorithms during the experiments. We first examine the impact of the attack duration on the attack results. We use $LCM(T_o,T_v)$ as an unit of the attack duration to evaluate DyPS since the offset between the observer task and the victim task repeats every $LCM(T_o,T_v)$. A total of 6000 task sets are tested with the attack duration varying from

![Figure 8: The impact of the start time delays caused by the tasks in harmony with the observer task on error ratio. The tested 6000 task sets are generated with at least one task in harmony with the observer task in each task set. The X-axis is the ratio of the estimated start time delays introduced by the harmonic tasks and the observer task’s period and the Y-axis is the error ratio. Each task set is tested with run-time variations and without run-time variations. The results show that the impact of the start time delays becomes subtle in the presence of run-time variations.

![Figure 9: The impact of the error ratio on the inference precision. Each error ratio bin is the result of the 6000 task sets (with harmonic tasks) with synthetically generated $\phi_v$ from $\{\phi_v + (0.01x)T_o \mid 0 \leq x \leq 10 \land x \in \mathbb{Z}\}$. It shows that the error ratio has negative impact on the inference precision. 10 \cdot LCM(T_o,T_v)). It is worth noting that only 6.75% of the task sets have constant delay and $E^o > 0$ since the task phases are randomly generated and hence it is not guaranteed that the harmonic tasks can always interfere with the observer task’s arrivals. This also indicates that having harmonic tasks does not necessarily downgrade the proposed attack. On the other hand, with execution time variations, the impact of the constant start time delay is significantly reduced due to the varied execution times. The DyPS algorithms yield better error ratios in all of the 6000 task sets (88.4% of the task sets that have $E^o > 0$ without run-time variations yield $E^o = 0$ with run-time variations.)

To understand the impact of the error ratio of $\hat{\phi}_v$ on the inference precision of $\hat{\phi}_v$, we test the aforementioned 6000 task sets with synthetically generated $\phi_v$ from $\{\phi_v + (0.01x)T_o \mid 0 \leq x \leq 10 \land x \in \mathbb{Z}\}$. A total of 6000 task sets are generated with at least one task in harmony with the observer task in each task set. The X-axis is the ratio of the estimated start time delays introduced by the harmonic tasks and the observer task’s period $T_o$. Each task set is tested with two conditions: (i) without run-time variations and (ii) with run-time variations. It is to examine the impact of the constant delay caused by the harmonic tasks in both a static and a more realistic RTS environment. Without run-time variations, the task’s execution times will run up to the WCET in every job instance which should retain the constant delay contributed by the harmonic tasks (or any potential delay contributed by other tasks). In contrast, as introduced in Section V-B with run-time variations, the task’s execution times include variations that are drawn from a normal distribution that gives a more realistic run-time result. The results are plotted in Figure 8 and show that, without run-time variations, the error ratio $E^o$ is proportional to the ratio of the constant delay caused by the harmonic tasks and the observer task’s period $T_o$. The outliers for the triangular points are the cases when there are other tasks contributing to the start time delays within the tested attack duration (i.e.,
Figure 10: The inference precision with varying attack duration. The X-axis is the length of the attacks based on \( LCM(T_o, T_v) \). It shows that the longer the attack persists the higher inference precision DyPS can achieve.

Figure 11: The impact of the number of tasks in a task set and the task set utilization on the inference precision. Each grid displays the mean inference precision for the corresponding number of tasks (Y-axis) and task utilization (X-axis). A darker grid shows a higher inference precision.

\( LCM(T_o, T_v) \) to \( 10 \cdot LCM(T_o, T_v) \) and the results are plotted in Figure 10. As shown, a longer attack duration leads to a higher inference precision. It is because more execution intervals are reconstructed as the attack lasts longer. The inference precision reaches \( T_v^o = 0.978 \) at \( 10 \cdot LCM(T_o, T_v) \) and plateaus afterward. Therefore, we choose to use an attack duration of \( 10 \cdot LCM(T_o, T_v) \) for the experiments presented in this section unless otherwise stated.

Next we break down the number of tasks in a task set and the task utilization to evaluate their impact on the inference precision. The experiment results are plotted in Figure 11. Each grid in the figure shows the mean inference precision of 100 task sets with the corresponding number of tasks in a task set and the task utilization. A brighter grid has a lower inference precision while a darker grid indicates that the attack yields a better inference precision. The resulting heat map gives an intuition for the distribution of the inference precision with varying number of tasks and the task utilization. The figure shows a small degradation when the number of tasks in a task set is high and the utilization is low. It is because a task set with a higher number of tasks can have more tasks preemption and delaying the observer task’s execution and this introduces more perturbations to the algorithms. On the other hand, a low utilization implies a low execution time for the observer task which results in shorter execution intervals to be reconstructed. This fact makes it difficult to effectively eliminate false time columns and leads to more scattered candidate time slots for the last step of the algorithms, which causes more uncertainty to the inferences.

3) Impact of DyPS Coverage Ratio: The DyPS coverage ratio \( C_{DyPS} \) represents the proportion of the time columns in a schedule ladder diagram that can be covered by the observer task’s execution. In the previous experiments with \( C_{DyPS} \geq 1 \), we showed that the DyPS algorithms are able to yield competitive inference precisions in various task set conditions. Here, we evaluate the case when the DyPS coverage ratio is less than one (i.e., \( 0 < C_{DyPS} < 1 \)). In this experiment, we generate 6000 task sets for each of the DyPS coverage ratio groups from \( \{0.001+0.1 \cdot x, 0.1+0.1 \cdot x \} | 0 \leq x \leq 9/3 \in \mathbb{Z} \} \). The mean inference precision is then taken from each DyPS coverage ratio group and the results are plotted in Figure 12. It shows that the DyPS algorithms get worse results as the DyPS coverage ratio drops. When \( C_{DyPS} \approx 0 \), the DyPS algorithms yield a inference precision \( T_v^o = 0.61 \) that is close to, but still better than, a random guess. Conversely, a higher DyPS coverage ratio has better inference precision since the observer task can cover more time columns on the schedule ladder diagram and hence has a higher chance to encapsulate the true time column.

4) Comparison with State-of-the-art (ScheduLeak): The main challenge in attacking the EDF RTS is the existence of the invalid part of the execution intervals. To understand the impact on the state-of-the-art (i.e., ScheduLeak) algorithms and how the DyPS algorithms handle such intervals, we test both algorithms in the EDF RTS with 6000 newly generated task sets (with \( C_o > (T_o - T_v) \), task utilization drawn from \((0.001, 1.0)\)) in which the presence of the invalid execution intervals are reconstructed as the attack lasts longer. The inference precision reaches \( T_v^o = 0.978 \) at \( 10 \cdot LCM(T_o, T_v) \) and plateaus afterward. Therefore, we choose to use an attack duration of \( 10 \cdot LCM(T_o, T_v) \) for the experiments presented in this section unless otherwise stated.

Figure 12: The inference precision with varying the DyPS coverage ratio in the range of \( 0 < C_{DyPS} < 1 \). It suggests that the DyPS algorithms get better performance as the DyPS coverage ratio increases. The DyPS algorithms perform better than a random guess when \( C_{DyPS} \approx 0 \).
intervals is guaranteed. Results presented in Figure 4 show that the DyPS algorithms outperform the ScheduLeak algorithms in handling the invalid execution intervals. As the attack lasts longer, more invalid execution intervals are collected in the case of ScheduLeak that leads to worse inference precision. The ScheduLeak algorithms yield a mean inference precision of 0.54 at 10 \cdot LCM(T_o, T_v). This is only slightly better than a naive attack with random guesses indicating ScheduLeak is inadequate for attacking the EDF RTS.

VI. DISCUSSION

From the evaluation results, DyPS attacks perform well in inferring the victim task’s phases and have similar performance characteristics to ScheduLeak 12 under various task set conditions. As a result, given the same task set, running either FP scheduling or EDF scheduling does not seem to make a significant difference in terms of resisting the attacks. However, the DyPS algorithms do require an extra step to reconstruct the observer task’s phase before proceeding to inferring the victim task’s phase, which is the crucial point that distinguishes the scheduler side-channels in FP RTS and EDF RTS. As analyzed in Section V-A, the observer task may suffer constant start time delays when there exist some harmonic tasks that have \( \psi(\tau_o, \tau_i) = 1 \) and \( \phi_i \) aligned with \( \phi_o \). Having such tasks may lead to an inaccurate \( \phi_o \) and further reduce the inference precision as evaluated in Section V-C.1. This offers one potential effective measure for defending EDF RTS against the DyPS attack. Consequently, employing the EDF scheduling algorithm and adjusting the task parameters to satisfy the aforementioned conditions can be a simple yet cost effective defense compared to the schedule obfuscation techniques (proposed for, e.g., the FP scheduling 9 and the time-triggered scheduling 10) that require fundamental changes in the design of the schedulers.

VII. RELATED WORK

Side-channels and covert-channels have been well studied in many research domains. Typical side-channels such as cache access time 19, power consumption traces 20, electromagnetic emanations 21, temperature 22, etc., may also be seen against RTS. In this paper, we focus on scheduler-based channels. Völpi et al. 23 examined covert channels between tasks with different priorities in the FP RTS. The authors proposed to modify the scheduler to alter the task that could potentially leak information with the idle task to avoid such covert channels. Kadloor et al. 24 presented a methodology for quantifying side-channel leakage for first-come-first-serve and time-division-multiple-access schedulers. Gong and Kiyavash 25 analyzed the deterministic work-conserving schedulers and discovered a lower bound for the total information leakage. There also has been other work on covert channels and side-channels in RTS 26, 27, 28, 29. In contrast to the above side-channels and the existing covert-channels, this paper focuses on the scheduler side-channels in which an unprivileged task is able to learn the behaviors of others by simply analyzing its own execution.

There exists a line of work that focus on the scheduler side-channels in RTS. Chen et al. 12 first introduced the scheduler side-channels in preemptive FP RTS and proposed the ScheduLeak algorithms that can extract valuable task information from the system schedules at run-time with using an unprivileged user-space task. Liu et al. 13 proposed a method to obtain the task parameters (i.e., the victim task’s period) that are required in the scheduler side-channel attacks. Yoon et al. 9 attempted to close the scheduler side-channels by introducing a randomization protocol that obfuscates the schedules in the FP RTS. Based on this idea, Krüger et al. 10 proposed a combined online/offline randomization scheme to reduce determinisms for time-triggered systems. Nasri et al. 11 conducted a comprehensive study on the schedule randomization protocol and argued that such techniques can expose the FP RTS to more risks. While these work are centered in the problem of the scheduler side-channels, none of them focused on the dynamic-priority RTS and thus the above defense techniques are inapplicable to preventing the proposed DyPS attack.

There has been some work on security integration in RTS 30, 31, 32, 33, 34, 35. Most of the work focused on defense techniques against general attacks. Some researchers framed security in RTS as a scheduling problem 36, 37. Hasan et al. 38 discussed the considerations of scheduling security tasks in legacy RTS. This is further extended to integrating security tasks into more general RTS models 39, 40. There exist another group of works focused on hardening RTS from the architecture perspective. Mohan et al. 41 proposed to use a disjoint, trusted hardware component (i.e., FPGA) to monitor the behavior of a real-time program running on an untrustworthy RTS. Yoon et al. 42 created the SecureCore framework that utilizes one of the cores in a multi-core processor as a trusted entity to carry out various security checks for the activities observed from other cores. Abdi et al. 43, 44 developed a restart-based approach that uses a root-of-trust (i.e., a piece of hardware circuit) and the trust zone technology to enforce the system reboot process to evict any malicious dwellers when necessary. While some of the techniques are useful for detecting anomalies and mitigating the impact of the attacks, they do not close the scheduler side-channels presented in this paper.

VIII. CONCLUSION

Dynamic priority scheduling algorithms such as EDF are able to better optimize the resources in the system (as compared to FP schedulers). In addition, they also increase the “dynamicism” in the system that can increase the difficulty for would-be-attackers looking to leak critical information via side-channels. Though not immune from such attacks, employing harmonic tasks with certain characteristics seems to have the potential to increase the difficulty for attackers. Hence, an increased adoption of dynamic scheduler, along with the aforementioned design choices, in RTS may aid in improving the security posture of such systems.
