Received August 3, 2020, accepted August 13, 2020, date of publication August 25, 2020, date of current version September 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019323

Task Parameters Analysis in Schedule-Based Timing Side-Channel Attack

SONGRAN LIU[1,2] AND WANG YI[1]
[1]Department of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
[2]Department of Computing, The Hong Kong Polytechnic University, Hong Kong

Corresponding author: Songran Liu (liusongran@stumail.neu.edu.cn)

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61532007.

ABSTRACT Recent work has shown that the timing behavior of a real-time system can be utilized by attackers for various adverse purposes via schedule-based timing side-channel attacks. An important assumption in this type of attacks is the prior knowledge of attackers about the task parameter information, including the number of tasks in the system and the period and execution time of each task. The attackers can use such information, together with the execution sequence of the task system, to reconstruct the exact schedule of the tasks and perform various subsequent attacks. In this paper, we show that the schedule-based timing side channel attacks can actually be performed even without knowing the task parameter information in prior. We develop methods to infer the number of tasks in the system and the period and execution time of each task directly from the execution sequence. This removes the task parameter prior knowledge requirement of the attackers and shows greater threats of the schedule-based timing side channel attacks. Both simulation experiments with synthetic task sets and a Zedboard-based evaluation with control system for a 3DOF helicopter are conducted to evaluate the proposed task parameter analysis method.

INDEX TERMS Real-time systems, side-channel attacks, task parameters analysis, signal analysis.

I. INTRODUCTION

A significant trend in computing systems is that they are more and more closely integrated into the surrounding physical environment, resulting in an emerging application paradigm, cyber-physical systems (CPS). Mis-operations of a cyber-physical system may lead to serious consequences, e.g., damage of devices and even loss of human lives. Therefore, cyber-physical systems must be designed with high security insurance. On the other hand, the real-time capability is a typical requirement in many cyber-physical systems, where the computing results must be delivered timely to match the inherent dynamics of the physical objects.

In traditional design methods for embedded real-time systems, the real-time capability and security are treated as two orthogonal issues. However, recent work showed that these issues in many cases are related [1]–[7], [9] and [10]. For example, a recent work [8] identifies a new type of side-channel attack to real-time systems, the schedule-based timing side-channel attack, where the exact *schedule* of a periodic task system can be reconstructed by analyzing the *execution sequence* (the *schedule* contains information of which task executes at each time point, while the execution sequence contains information of whether the CPU is idle or not at each time point), as depicted in Figure 1. After accomplishing the reconstruction of task schedule, the attackers could pinpoint the arrival instant and start instant of any task in the victim system, thereby carrying out some further attacks. We should note that, the precise schedule information can assist attackers to filter out more noise, thus leading the attack harder to be detected. In the cache-based side-channel attack proposed in [8], the target of attackers is to observe the cache usage of an image encoding task in a UAV system. The results (Figure 13(a) in [8]) show that, with assistance of the schedule information the attacker can effectively catch the cache usage behavior of the victim task compared with the ones with no such information. As another example in a stealthy attack against robotic vehicles [12], the precondition of such attacks is to insert artificial sensor data just before the control task execution.

An important assumption in [8] is that the task parameters are known in prior by the attackers. More specifically, it assumed that the attackers know information about the number of tasks in the system, the period and execution time

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

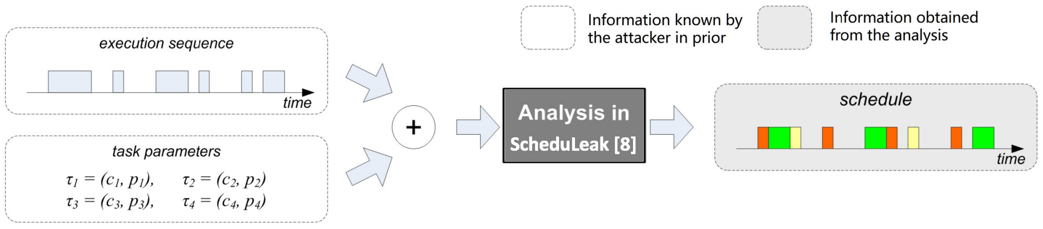

Information known by the attacker in prior
Information obtained from the analysis
execution sequence
time
task parameters
$\tau_1 = (c_1, p_1)$, $\tau_2 = (c_2, p_2)$
$\tau_3 = (c_3, p_3)$, $\tau_4 = (c_4, p_4)$
+
Analysis in ScheduLeak [8]
schedule
time

FIGURE 1. The analysis flow of the schedule-based timing side channel attack proposed in [8].

of each task. The authors argued that the above information can be obtained by social engineering or hacking into task scheduler of the real-time operating system at run-time. Although this may be possible in some cases, in general, obtaining such information may not be easy. For example, to get such information from the task scheduler requires the attackers to hack into the operating system kernel, which is usually protected by high-level security mechanisms.

Side-channels and covert-channels have been well studied in many research domains. Typical side-channels such as cache access time [26], power consumption traces [27], electromagnetic emanations [25], temperature [28], etc., may also be seen against real-time system (RTS). In this paper, we focus on scheduler-based channels. Son *et al.* [29] demonstrated that the covert channel will not be closed completely due to the RM scheduling. Then they proposed a modified scheduler that use idle thread to replace the unsafe threads [30]. They also considered the share-resource covert channels and gave a kind of transformed resource locking protocols [31]. Kadloor *et al.* [32] considered the timing side channel attack in the context switch of shared scheduler and two feasible schedulers (first-come-first-serve and time-division-multiple-access) are given to solve this kind of problem. There also has been other work on covert channels and side-channels in real-time system [31], [33], [34]. In contrast to the above side-channels and the existing covert-channels, this paper focuses on the study of scheduler-based side-channels which can assist stealthy attacks.

There exists a line of work that focus on the scheduler side-channels in RTS. Chen *et al.* [8] first introduced the scheduler side-channels in preemptive fixed-priority RTS and proposed the *ScheduLeak* algorithms that can infer the exact execution information of each task from the system schedules at run-time with using an unprivileged user-space task. But a fundamental assumption is that the adversary has full knowledge of the real-time task set information (e.g., period, execution time), which reduces the threat and practicality of such a schedule-based side-channel attack. Our work aims to fill this gap. Liu *et al.* [35] proposed a method to trace the engine speed variation based on information leakage through scheduler side-channels. However, their method is hard to fit into a static task model considered in this work. Yoon *et al.* [36] and [37] attempted to close the scheduler side-channels by introducing a randomization protocol that obfuscates the schedules in the FP RTS. Based on this idea, Kruger *et al.* [38] proposed a combined online\offline randomization scheme to reduce deterministic for time-triggered systems. Nasri *et al.* [39] conducted a comprehensive study on the schedule randomization protocol and argued that such techniques can expose the FP RTS to more risks. Nevertheless, these solutions are not applicable to most real-time systems in which a preemptive, fixed-priority scheduler supports both periodic and sporadic real-time tasks. This leaves those systems still vulnerable to schedule-based side-channel attack.

Another class of work is about utilizing timing property of a RTS to assist system testing and validation. Iegorov *et al.* in [40] use scheduling sequences to assist the understanding of runtime behavior of the a RTS. They utilize the sequences to classify all the tasks into two categories: periodic task or sporadic task, then estimates the period value and execution time interval of each periodic task. However, similar to the work in ScheduLeak [8], the prior knowledge of their work are, the exact time point of each task context switch and the task ID (or thread ID) of each sub-busy interval. While we remove these assumptions in this work, all we need in our method is a pure scheduling sequence composed of busy interval and idle interval.

The integration of security into real-time schedulers is a developing area of research. Most of the work focused on defence techniques against general attacks. Mohan *et al.* [41] offered a consideration of real-time system security requirements as a set of scheduling constraints and introduced a modified fixed-priority scheduling algorithm that integrates security levels into scheduling decisions. There exist another group of works focused on hardening RTS from the architecture perspective. Yoon *et al.et al.* [10] 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.* [42], [43] 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 discussed in this paper.

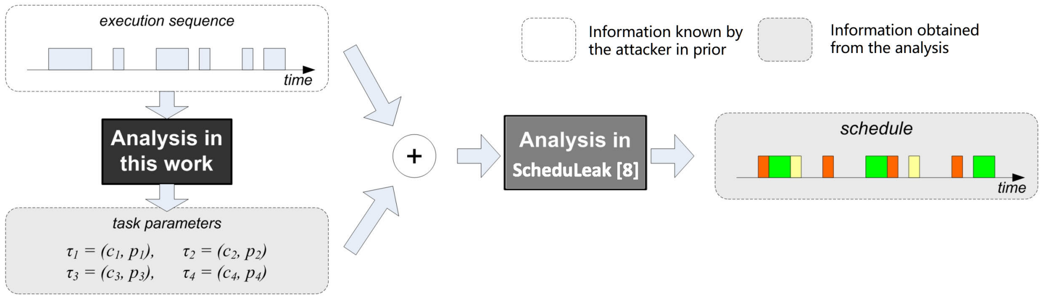

execution sequence
time
Analysis in this work
task parameters
$\tau_1 = (c_1, p_1)$, $\tau_2 = (c_2, p_2)$
$\tau_3 = (c_3, p_3)$, $\tau_4 = (c_4, p_4)$
+
Analysis in ScheduLeak [8]
Information known by the attacker in prior
Information obtained from the analysis
schedule
time

FIGURE 2. The analysis flow of the schedule-based timing side channel attack using the task parameter analysis proposed in this work.

In this paper, we will show that actually the schedule-based timing side-channel attack can be done even without prior knowledge about the task parameters. Instead, such information can be obtained from the execution sequence, as shown in Figure 2. The technical contribution of this work is developing methods to extract such task parameter information (number of tasks and the execution time and period of each task) from the execution sequence.

Note that it is not always possible to correctly infer task parameters from the execution sequence. Consider an extreme case where the total utilization of a periodic task set is 1, and thus the processor is always busy. Such an execution sequence does not provide any useful information about each individual task. Even for task sets with utilization smaller than 1, there are cases where different task parameters leading to exactly the same execution sequences and thus it is impossible to distinguish among the different possible task parameters. Therefore, the target of this paper is not to design a method that guarantees to always correctly derive the task parameter information from execution sequence (this is indeed impossible as witnessed by the above examples). Instead, our target is to design a method that can successfully do this for as many task sets as possible. The experimental evaluation in Section IV shows that our method can correctly obtain the desired information in most cases. The result of this paper shows that in a large portion of cases the attacker can successfully launch the schedule-based timing side-channel attack without knowing the task parameter information, and the threat of the schedule-based timing side-channel attack is actually even greater than that is stated in [8].

Obtaining the task parameters from the execution sequence is technically challenging. This paper developed a two-step method to solve this problem: (1) select a set of possible candidates for task periods and (2) derive the computation time for each task corresponding to each period candidate. The first step is done by analyzing the spectrum features of the execution sequence. The second step is by modeling the target problem as a mixed-integer nonlinear programming (MINLP) problem. If the resulting execution time corresponding to a period candidate is zero, then we know the task set does not contain a task with this period value, and thus the number of tasks in the system is also clear. We conduct both experiments with synthetic task sets and a control system of a 3DOF helicopter to evaluate the proposed task parameter analysis methods.

II. MODEL

A. SYSTEM MODEL

We consider a task system composed of a set of N periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots \tau_N\}$. Each task τ_i, $1 \leq i \leq N$, is characterized by a tuple $\{c_i, p_i\}$, in which c_i is the execution time and p_i is the period. The tasks have implicit deadlines, i.e., p_i is also the relative deadline of task τ_i.

We assume the system is scheduled by the Rate Monotonic (RM) scheduling algorithm [11], which is the most widely used scheduling algorithm in industrial real-time systems. In RM, task priorities are decided by their periods: the smaller period, the higher priority. We further assume each task has a unique period, and consequently a unique priority. A task is schedulable if all of its released instances can finish execution before the next release time, and a task set is schedulable if all of its tasks are schedulable.

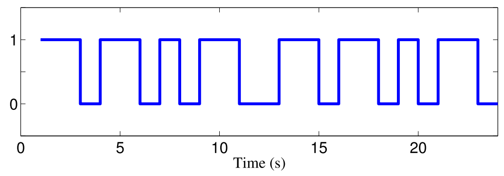


FIGURE 3. A piece of scheduling sequence of length 24.

B. ADVERSARY AND ATTACK MODEL

As shown in Figure 2, we assume the attacker has obtained a sufficiently long execution sequence of the task system. The execution sequence may be any arbitrary segment of the whole execution trace. The execution sequence is represented by a binary sequence x. Each binary $x(n)$ represents whether the processor is busy or idle at the n^{th} time unit. For example, Figure 3 shows an execution sequence example, the binary representation of which is 110110101100110110101110. There are many ways to obtain the scheduling sequences, including software-based approaches such as hacking into the idle task and hardware-based approaches such as monitoring the electromagnetic radiation of the processor or the signal of the wires or pins. In Section IV-A we will discuss how to obtain the execution sequence in detail.

The target of the attacker is to obtain the following task parameter information from the execution sequence:

- the number of tasks in the system
- the execution time c_i and period p_i of each task τ_i

After obtaining the above information, the execution sequence and the task parameter information can be combined to reconstruct the schedule by the method in [8], and perform subsequent adversarial attacks as discussed in [8].

III. METHOD

This section presents methods to obtain the task parameters from the execution sequence. The method consists of two major steps:

1) Identify a set of candidate values for task periods. We first compute the hyper-period of all tasks, which provides important information to limit the range of possible task periods. Then we present two methods to select candidate values for task periods. The first one is based on the Discrete Fourier Transform (DFT), and the second one is our new design that guarantees to include all the task periods in the candidate set.
2) Infer c_i of each task τ_i. This is modeled as a series of optimization problems, each giving c_i for the task corresponding to one task period candidate p_i ($c_i = 0$ implies there is no task with this candidate period in the task system).

We design such a two-step strategy for better scalability. If we also consider the number of tasks and task periods as unknown variables and model everything into a unified MINLP problem, the search space will be significantly larger. In the following sections we will introduce each step in detail.

A. SELECT PERIOD CANDIDATES

First of all, we determine the hyper-period of all the tasks in the system, denoted by H. Since the tasks are periodic, the execution sequence repeats in every hyper-period. We will use Circular Autocorrelation based method [13], a widely used technique in signal processing, to decide the hyper-period of all the tasks.

The circular autocorrelation $ACC(\phi)$ is a metric to describe that how similar a sequence is to its ϕ circular shift:

$$ACC(\phi) = \frac{1}{N}\sum_{n=0}^{N-1} x(\phi)x(n+\phi) \tag{1}$$

where $x()$ is the binary execution sequence. Since the execution sequence repeats in every hyper-period, the value of $ACC(\phi)$ with ϕ being the multiple of the hyper-period is clearly larger than with other values of ϕ. Figure 4 shows the resulting $ACC()$ of an execution sequence where the hyper-period is 45. We can see that the value $ACC(\phi)$ reaches a peak when ϕ is the multiple of the hyper-period 45.

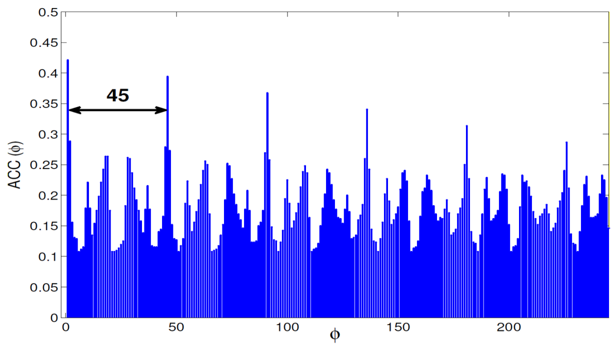


FIGURE 4. ACC result of a 520-length scheduling sequence generated by task set {(2, 9), (3, 15)}.

The computation of ACC can be realized by *inverse discrete fourier transformation* with a time complexity of $O(NlogN)$ (N is the length of the execution sequence x) [14]:

$$ACC = \mathcal{DFT}^{-1}(x \otimes x^T) \tag{2}$$

where x^T is the complex conjugation of x and $\otimes$ denotes the inner product operation. The details about $\mathcal{DFT}^{-1}$ are provided in the appendix.

Since the execution sequence repeats with the period H, we only need to consider a subsequence of the execution sequence with length H. Therefore, for the sake of simplicity, we use x to represent a subsequence of the original execution sequence in the remaining of this section.

The hyper-period obtained above is useful for selecting the candidates for task periods: only the divisors of the hyper-period are eligible to be task periods. However, this may result in a large number of period candidates. In the following, we present two methods to further select period candidates among the divisors of the hyper-period. The first method uses the classical discrete fourier transform (DFT), and the second method is our new method that outperforms the first method.

1) THE FIRST METHOD

The execution sequence can be viewed as a kind of combination of multiple sub-sequences, each sub-sequence corresponding to the released workload (i.e. released job) of one periodic task. Our target is to analyze the period of the sub-sequences. This is similar to the frequency of signal sequences in signal processing. Any signal wave sequence can be reconstructed as the summation of sinusoids with different frequencies. The Discrete Fourier Transform (DFT) is used to analyze the amplitude of the sinusoid of each frequency for each discrete signal sequences. The details about DFT are provided in the appendix.

Inspired by the similarity between our problem and the frequency analysis of discrete signal sequences, we use DFT to transfer the execution sequence into the frequency domain, with which we can select the periods corresponding frequency components with larger amplitudes as the desired candidates. We set a threshold on the amplitude to select the candidates. An overly large threshold may exclude real period from our consideration, while an overly small threshold may include too many false candidates so that the following steps will be more difficult to solve. In this subsection, after our experimental comparison, we assume the threshold to be 0.05.

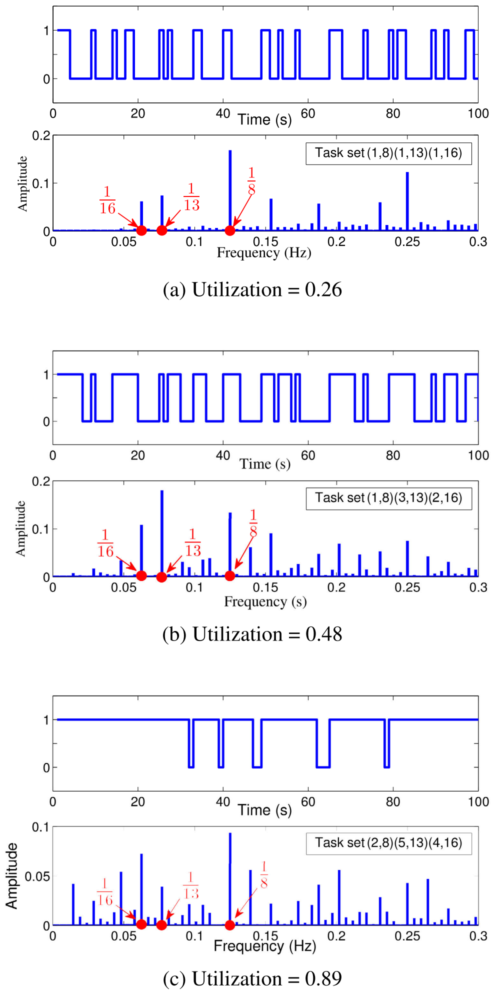


FIGURE 5. **DFT results of task sets in different utilization.**

Figure 5 presents the resulting amplitude spectrum of several execution sequences. The upper figure in Figure 5-(a) is an execution sequence of three tasks with total utilization of 0.26, and the lower figure is the resulting amplitude spectrum. We can see there are seven frequency components with significantly larger amplitude than others, among which three corresponds to task periods. However, in this example, the task period do not all correspond to the frequency components with the largest amplitudes: the second largest frequency component actually does not correspond to any task period.

If we increase the execution time of the second and the third task and get a task set with larger total utilization (0.48), the accuracy of the analysis is even better: the three largest frequency components exactly correspond to the three task period, as shown in Figure 5-(b).

If we further increase the execution time (and thus the total utilization), the accuracy will drop again. Figure 5-(c) shows the execution sequence and the amplitude spectrum of a task set with total utilization of 0.89. This time the amplitude of the frequency component corresponding to period 13 is smaller than other noisy frequencies.

In summary, the amplitude spectrum obtained by DFT indeed provides useful information about possible periods, but these information are not always precise. In general, it works better for task sets with medium utilization, but relatively worse with small or large utilization (which will be discussed in detail in Section III-B).

There are two major reasons why the amplitude spectrum may not correctly reflect the task periods. First, while in a signal wave is a direct summation of component sinusoids, the binary execution sequence is *not* a direct summation of the released workload but a combination with shifting. For example, if two tasks both release one unit workload at time t, $x(t)$ is 1 not 2 and one of them is shifted to be executed at time $t+1$, i.e., reflected by $x(t+1) = 1$. Second, DFT uses sinusoids to fit the square waves in the binary execution sequence, which may incur significant errors. Figure 6 shows the best fit of a square wave sequence with sinusoids, where the red line is the summation of the sinusoids. We can see that the square waves are only approximately fit by the sinusoids, with many extra noisy sinusoids that correspond to neither of the two tasks. In general, there is no provable quality guarantee of this DFT-based method. It sometimes includes a significant number of useless candidates, but sometimes may contain real task periods.

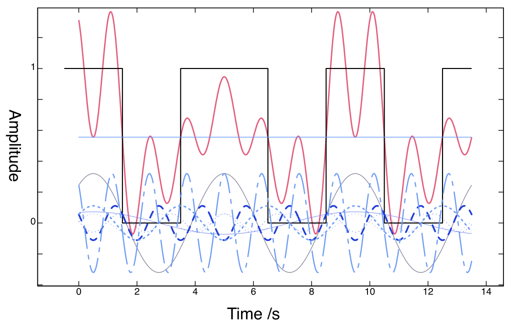


FIGURE 6. **A series of sinusoids generated by DFT to task set {(2, 4), (1, 9)}.**

2) THE SECOND METHOD

Now we present a new method that can address the shortcomings of the first method based on DFT and thus better select period candidates. The complexity of the second method is higher, but guarantees to include all task periods in the candidate set. We first discuss the rationale behind the proposed method, and then exemplify the workflow of Algorithm 1.

The pseudo-code of our new period candidate selection algorithm is shown in Algorithm 1. x is the execution sequence of length H (recall that we only need to consider a sub-sequence of the original execution sequence with length H). We use L_{max} to indicate the maximal length of continuous idle interval of the execution sequence, i.e., the maximal number of consecutive 0 in x. First, we can draw the following lemma about L_{max}:

Lemma 1: In a feasible scheduling sequence of a periodic task set, the maximal length of idle interval of the execution sequence, denoted by L_{max}, is smaller than the period of any task.

Proof: Suppose a task τ_i's period $p_i \leq L_{max}$. Without loss of generality, we assume $[t, t + L_{max} - 1]$ is an idle interval. Since $p_i \leq L_{max}$, there exists at least one time point t' in $[t, t+L_{max}-2]$ at which τ_i releases a job but the processor is idle during the following time unit $[t', t' + 1)]$, which contradicts the work-conserving property of fixed-priority scheduling. □

By the above lemma we know the period candidates must be at least L_{max}. On the other hand, they must be below $H/2$ (except that H may also be a task period). Therefore, the algorithm will check each divisor of H in the range $[p, \ldots, H/2]$ and H to see if they are valid candidate for task periods (the for loop in line 3 of Algorithm 1).

To decide whether a value is a valid candidate, we apply function $\Omega(x, i)$ to binary sequences x and a number i, which is a divisor of $|x|$ (the length of x), as defined in the following:

$$\Omega(x, i) = x_1 \& x_2 \& \ldots \& x_{\frac{|x|}{i}} \tag{3}$$

where x_m is the m^{th} sub-sequence of x of length i, and & is the bit-wise AND operation of two binary sequences. For example, given a sequence $x = 111010100010$ and $i = 4$.

$$\Omega(x, 4) = 1110 \& 1010 \& 0110 = 0010 \tag{4}$$

Algorithm 1 Period Candidates Selection

```
  Input: execution sequence $x$ and hyper-period $H$.
  Output: period candidate set $\overline{P} = \{\overline{p_1}, \overline{p_2}, \ldots\}$.
1 $\overline{P} = \emptyset$
2 $p = L_{max} + 1$
3 for $\rho$ = each divisor of $H$ in $[p, \ldots, H/2]$ and $\rho = H$ do
4     if $\Omega(x, \rho)$ is not an all-zero sequence then
5         put $\rho$ in the period candidate set $\overline{P}$;
6     end
7 end
```

Theorem 1: The output $\overline{P}$ of Algorithm 1 includes the periods of all tasks in the system.

Proof: We consider an arbitrary task τ_i with period p_i in the system. First, p_i is a divisor of the hyper-period H, and in the range $[p, H/2]$ or is H as discussed above. So the algorithm executes the condition checking in line 4 with $\rho = p_i$.

We look into the $\Omega(x, p_i)$ operation. $\Omega(x, p_i)$ first divide x into several subsequences, each of length p_i. So we know a job of τ_i must be released exactly once in each subsequence with exactly the same offset relative to the beginning of each subsequence. When a job of τ_i is released, the following time unit must be busy (with a bit 1 in the binary sequence) due to the work-conserving property of fixed-priority scheduling. Therefore, the bit correspond to the job releases of τ_i in each subsequence must be 1, and thus $\Omega(x, p_i)$ must not be all-zero sequence. □

To illustrate how Algorithm 1 works, we use a scheduler sequence $x = 111010101000$, generated by task set $\{(1, 4), (1, 6), (1, 12)\}$. First, we have $L_{max} = 3$. Then, the available divisor set of 12 is $\{4, 6, 12\}$. After applying function $\Omega(x, 4)$, $\Omega(x, 6)$ and $\Omega(x, 12)$ to sequence x, the corresponding results are all positive. So divisor $\{4, 6, 12\}$ are all treated as period candidates.

B. DECIDE EXECUTION TIMES

In this section we will decide the execution time of each potential task corresponding to a period candidate. If the resulting execution time corresponds to some period candidate is 0, it means there is no task in the system having this period value, by which the number of tasks in the system is also decided.

The decision of execution times is modeled as a mixed integer non-linear programming (MINLP) problem, with the optimization target:

$$\text{Minimize} \quad \left(\sum_{i=1}^{M} c_i/\overline{p}_i - util\right)^2 \tag{5}$$

util is a known constant representing the total utilization of the task set obtained from the execution sequence, which equals the ratio between number of bits 1 in the execution sequence and the total length of the execution sequence (which is H). M is the size of $\overline{P}$, i.e., the number of period candidate. All period candidates $\overline{p_i}$ are known. Each c_i is an open variable representing the execution time of the potential task corresponding to period candidate p_i. The target of the MINLP problem is to find assignments of each c_i so that the total utilization of the task system calculated by task parameters is as close as possible to *util*.

There are three groups of constraints as listed in the following:

$$\forall i \in [1, M] : 0 \leq c_i \leq c_i^{max} \tag{6}$$

$$\forall i \in [1, M] : \phi_i^{min} \leq \phi_i \leq \phi_i^{max} \tag{7}$$

$$\forall s \in [1, S] : \sum_{i=1}^{M} \left\lceil \frac{\delta(s, i)}{\overline{p_i}} \right\rceil \times c_i = L_s \tag{8}$$

$$\text{where } \delta(s, i) = L_s + \alpha_s - \phi_i - \left\lceil \frac{\alpha_s - \phi_i}{\overline{p_i}} \right\rceil \times \overline{p_i} \tag{9}$$

In the following we explain each group of constraints.

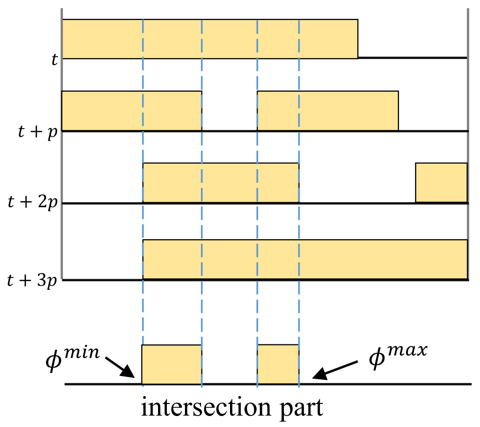


FIGURE 7. Example of reducing interval of offset $\overline{\phi_i}$.

1) EXECUTION TIME CONSTRAINTS

Constraint (6) restricts the range of c_i for each potential task i, where c_i^{max} is a constant pre-calculated by $\lfloor \overline{p_i} \times util \rfloor$.

2) OFFSET CONSTRAINTS

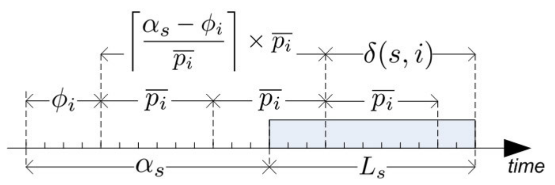
Each ϕ_i in is an auxiliary variable representing the offset of the first release of this task relative to the beginning of the considered execution sequence. Constraint (7) limits the range of ϕ_i of each task. A naive lower bound and upper bound of ϕ_i is 0 and $\overline{p_i}$, respectively. We can calculate tighter upper and lower bounds for ϕ_i using the $\Omega(x, \overline{p_i})$ function (introduced in last subsection). $\Omega(x, \overline{p_i})$ divide the execution sequence x into $H/\overline{p_i}$ subsequences, and returns the results of the bit-wise AND of all these subsequences. Only the locations of the 1 bits in this resulting subsequence can be possible values of ϕ_i, or alternatively, the the location of the first 1 and the last 1 in this resulting subsequence is the lower and upper bound of ϕ_i, respectively, as shown in Figure 7.

3) WORKLOAD CONSTRAINTS

The execution sequence contains multiple busy periods (the subsequence with all bits being 1). S is the number of busy periods in the execution sequence, L_s is the length of the s^{th} busy period, and α_s is the start time of the s^{th} busy period (relative to the beginning of the execution sequence).

The intuition of Constraint (8) is that the total workload released by all tasks in each busy period should be equal to the length of this busy period. $\delta(s, i)$ calculates the distance between the first release time of task i in the s^{th} busy period and the end of the s^{th} busy period, as illustrated in Figure 8. Therefore, $\lceil \delta(s, i)/\overline{p_i} \rceil \times c_i$ is the total amount of workload of task i in the s^{th} busy period, and the LHS of the equation in Constraint (8) is the total workload of all tasks to be executed in the s^{th} busy period, which equals the length of this busy period.

The ceiling operator in Constraint (8) can be handled by a simple technique: replacing $\lceil z \rceil = y$ (where z is a real number and y is an integer) by $y-1 < z \leq y$. Then the overall MINLP problem can be solved by common solvers such as SCIP [15], NOMAD [16] and CPLEX [17].



$\left\lceil \frac{\alpha_s - \phi_i}{\overline{p_i}} \right\rceil \times \overline{p_i}$
$\delta(s,i)$
ϕ_i
$\overline{p_i}$
$\overline{p_i}$
$\overline{p_i}$
α_s
L_s
time

FIGURE 8. Illustrating the calculation of $\delta(s, i)$.

IV. EVALUATION

In this section we first discuss how to obtain the execution sequence in realistic systems. In what follows, we empirically evaluate our proposed analysis method to infer the task parameters from the execution sequences. Finally we present a case study utilizing a Zedboard-based 3DOF helicopter control system.

A. OBTAIN THE EXECUTION SEQUENCES

Alternative 1: A software-based method was introduced in [8] to obtain execution sequences from realistic systems. More specifically, as most embedded systems(e.g., FreeRTOS) adopt a flat memory model (and no virtual memory or other protection mechanisms such as address-space layout randomization) [18], attackers could hijack the idle task of the operating system by exploiting memory corruption vulnerabilities(e.g., buffer overflow [19] or return-oriented programming [20]). For instance, by replacing the pointer of function `ApplicationIdleHook()` called by idle task in FreeRTOS with our malicious function which records the preemption points of idle task in the victim system, the adversary could obtain the execution sequences with the resolution of the *Global Timer*. Note that, as this paper aims to complement the work in [8], we capture the execution sequences utilizing above software-based method to keep consistency and fair comparison.

Alternative 2: When standard interfaces, e.g., network interfaces and debug interfaces [21] of the victim embedded system, are su?ciently protected or it is hard to hack into the software system, an non-intrusive approach is indispensable to capture the execution sequences. Actually, in order for a side-channel attack to be effective in practical scenarios for a security breach, it has to be applicable without having physical access to the device being attacked [22]. Electromagnetic (EM) side-channel is one approach that has shown promising results. It requires minimum physical manipulations to the device being inspected [23]. EM emissions of a device can be passively observed to infer both the internal operations being performed and the data being handled [24].

In what follows, we show it is possible to utilize an EM side-channel to obtain the execution sequence of a MSP430 board. This method[1] utilizes the spectrum difference

[1]As state before, we use the method in *Alterative 1* to keep consistency with the work [8]. And we will perfect the EM-based approach in our future work.

(a) Agilent N9030A signal analyzer (b) MSP430 board

FIGURE 9. The signal analyzer and hardware board.

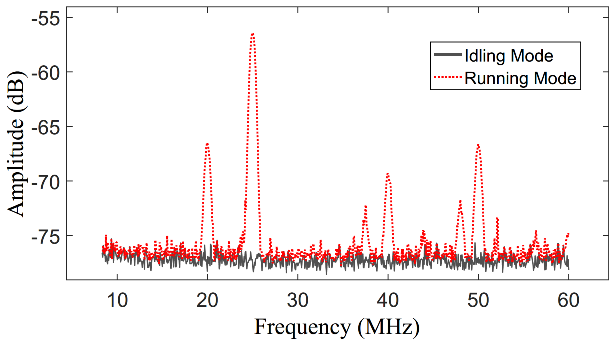


FIGURE 10. Electromagnetic radiation spectrum in the idling and running modes.

in the electromagnetic emissions of the hardware board between the running and idling modes. To illustrate the feasibility of this approach, we use the Agilent N9030A signal analyzer [44] (Figure 9-(a)) to record and analyze the electromagnetic radiation of an MSP430 (working at 25MHz) [45] hardware board (Figure 9-(b)). We program the MSP430 ECU to switch between the running and idle modes. Then we use ROHDE & SCHWARZH(HZ-15) near field probes [46] to measure the electromagnetic radiation in both modes, and use the Agilent N9030A signal analyzer to obtain the electromagnetic radiation spectrum in the running and idle modes. As shown in Figure 10, we can see that the spectrum of two modes are substantially different. More specifically, the EM signal at *Running mode* has larger frequency component at 25MHz and 50MHz when compared with its counterpart at *Idling mode*. Note that, 25MHz here corresponds to the main frequency of MSP430 MCU, while 50MHz is its harmonic. Therefore, it is easy to develop classification algorithms (e.g., neural network-based [25]) to automatically distinguish them, then constructing execution sequences.

B. SIMULATION-BASED EVALUATION

1) SIMULATION SETUP

The simulation experiments were used to test the scalability of our proposed methods, also to test with a more diverse set of real-time task combinations.

a: TASK SET GENERATION

We use the randomly generated synthetic task sets to evaluate the performance of the proposed analysis method under different settings. We generated 10000 schedulable synthetic task sets randomly and divided them into groups according to their total utilizations: the i^{th} group contains task sets with total utilization in the range of $[0.1i, 0.1i + 0.1)$. The number of tasks in each task set is randomly chosen in the range of [5, 15]. The task periods are randomly drawn from [100, 1000] and the execution time of a task is limited to be integers between 1 and half of its period, i.e., $c_i \in [1, p_i/2)$. We use the constraint solver SCIP [15], which can solve MINLP problem with up to hundreds of variables. The experiments are executed on a desktop PC with a dual-core Intel i5-2450M processor, with 2.5G frequency, 3M caches and 12G DDR3 1333 MHZ memory.

b: SCHEDULE ALGORITHM

We use the commonly used rate-monotonic algorithm [11] to assign the priorities of tasks, i.e., a task with a shorter period is assigned a higher priority. When generating task sets according to the above setting, we discard those are not schedulable by RM scheduling and only keep the schedulable ones for the experiments.

2) SIMULATION RESULTS

In Section III.A, we present two period candidate selection methods. The first method, denoted by *pcs-DFT*, is based on DFT as presented in Section III.A.1. The second method, denoted by *pcs-NEW*, is our new design in Section III.A.2 that overcomes the shortcomings of *pcs-DFT* and guarantees to include all the task periods in the candidate set. The capability of *pcs-DFT* to successfully include the task periods into the candidate set depends on the threshold value. If the threshold is too large, the possibility for *pcs-DFT* to exclude some real task periods from the candidate set $\overline{P}$ is high, and thus will lead to poor analysis precision. On the other hand, if the threshold is too low, too many negative candidates will be included in $\overline{P}$, which will lead to very large MINLP problems in the execution time decision phase. In the experiments of this subsection, we set the threshold to be 0.01 to limit the size of period candidate set so that the subsequent execution time decision problem can be solved in acceptable time (in 10 hours).

a: INFLUENCE OF TASK SET UTILIZATION

Figure 11 shows the analysis precision of our overall method with these two different period candidate selection methods. The x-axis of the figure represents different range of total utilizations of the generated task sets. The y-axis represents the analysis precision, in terms of the ratio between the number of task sets whose parameters are correctly obtained (the c_i and p_i for every task τ_i are correctly obtained). From Figure 11 we can see that *pcs-NEW* can achieve nearly 100% analysis precision except for task sets with large total utilizations. *pcs-DFT* also performs well for task sets with medium total utilizations, but the quality drops for task sets with large or small total utilizations.

In general, the task sets with large total utilizations are difficult to analyze since the processor is in most time busy

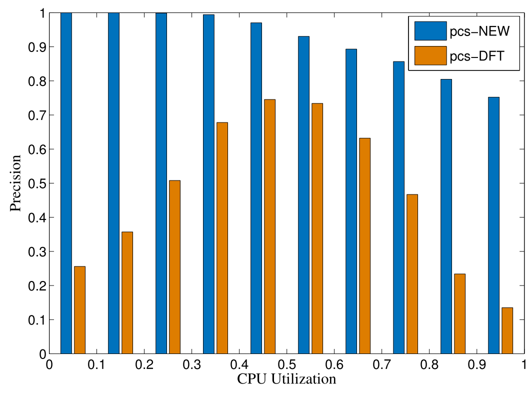
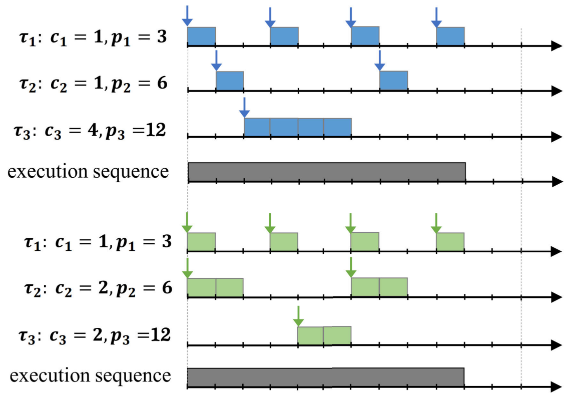


FIGURE 11. Precision comparison between *pcs-DFT* and *pcs-NEW*.

and the periodic pattern is less obvious. An extreme case is a task set with total utilization 1, for which the execution sequence are all 1 and it is impossible to infer the task parameters. Figure 12 shows another example where two task sets with different parameters results in exactly the same execution sequence, and thus no method can guarantee to correctly infer the task parameters. In general, the task sets with large total utilizations are inherently more difficult to analyze, and thus both *pcs-DFT* and *pcs-NEW* perform worst with them.



τ_1: $c_1 = 1, p_1 = 3$
τ_2: $c_2 = 1, p_2 = 6$
τ_3: $c_3 = 4, p_3 = 12$
execution sequence
τ_1: $c_1 = 1, p_1 = 3$
τ_2: $c_2 = 2, p_2 = 6$
τ_3: $c_3 = 2, p_3 = 12$
execution sequence

FIGURE 12. An example to show two different task sets (with total utilization smaller than 1) can lead to exactly the same execution sequence.

The *pcs-DFT* method also performs worse for task sets with small total utilizations. As discussed in Section III.A.1, the principle of DFT is to use sinusoids to fit the square waves. Due to the symmetry of sinusoids, there is no difference between using sinusoids to fit a sequence of square waves and its dual sequence (obtained by replacing all 1 by 0 and also the other way around). The dual sequences of the execution sequences of task sets with low total utilization can are similar to the execution sequences of task sets with high utilizations. Therefore, for the *pcs-DFT* method the task sets with low total utilization are equally difficult to analyze.

Although the performance of *pcs-DFT* in terms of analysis precision is in general worse than *pcs-NEW*, the efficiency

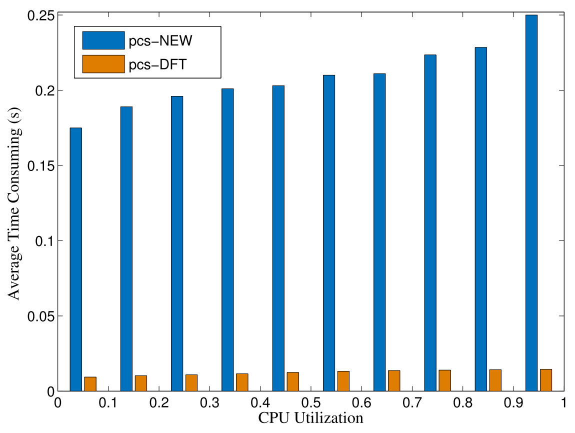


FIGURE 13. Time consumption comparison between *pcs-DFT* and *pcs-NEW*.

of *pcs-DFT* is higher than *pcs-NEW*. Figure 13 shows the comparison of the running time of *pcs-DFT* and *pcs-NEW*. Nevertheless, as will be shown later, the step to decide execution times (by solving the MINLP problems) is the real bottleneck of the efficiency of the overall approach. Comparing with the running time to solve the MINLP problems, the overhead incurred by both *pcs-DFT* and *pcs-NEW* are neglectable. Therefore, in the following experiments we use *pcs-NEW* to select the period candidates.

b: RECOGNITION OF LOW-UTILIZATION TASK

We also want to know the recognition ratio of a single task with different proportion of utilization in total task set. Intuitively, the less utilization occupied of a individual task, the harder it can be recognized. Figure 14 shows analysis precision of our approach in different ranges of individual task utilization. The x-axis represents the range of task utilization, and the y-axis represents the analysis precision, in terms of the ratio between the number of tasks whose parameters are correctly obtained and the total number of tasks in this utilization range. From 14, we can see that tasks with lower utilization are more difficult to be recognized.

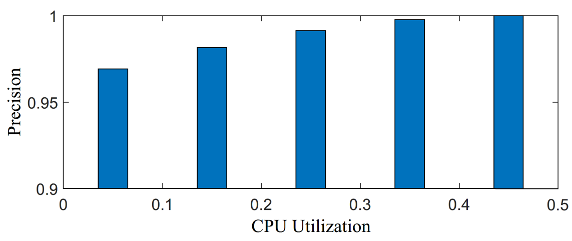
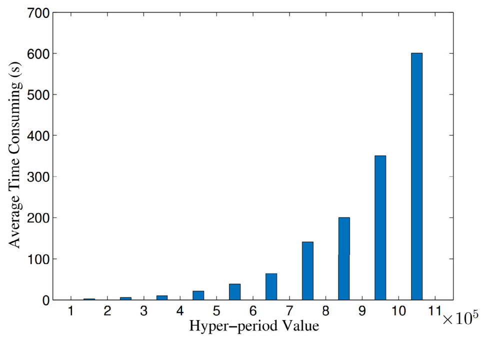


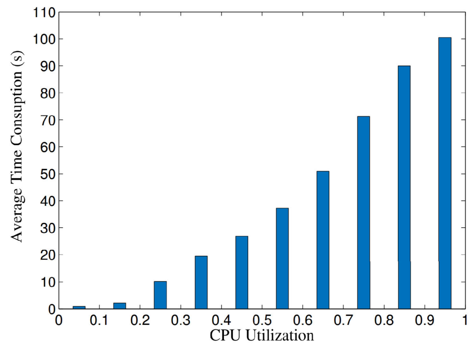
FIGURE 14. Analysis precision of individual tasks with different ranges of utilization.

c: ANALYSIS OF CALCULATION TIME

In the following we evaluate the analysis efficiency of the overall approach. In Figure 15-(a), the x-axis represents the range of hyper-periods of the task sets, and the y-axis is the average time consumption to perform the overall task parameter analysis, including both the first step *pcs-NEW* that selects period candidates and the second step using the solver SCIP to solve the MINLP problem to determine the execution times. As we can see from Figure 15-(a), the time consumption increases exponentially with respect to the hyper-periods. Figure 15-(b) shows the average time consumption increases (almost linearly) with respect to the total utilization.



(a) Time consumption for task sets with different ranges of hyper-period.



(b) Time consumption for task sets with different ranges of total utilization.

FIGURE 15. Time consumption for task sets with different ranges of hyper-period and utilization.

The above results indicate that our proposed task parameter analysis method can be quite expensive, especially when the task set has a large parameter scale. Therefore, our method is only suitable for attacks that are not performed in real-time, i.e., the attacker first obtains the execution sequence, and then spends time to analyze the task parameters, after which the task parameter information can be used to launch subsequent adversary actions. With our proposed task parameter analysis method, the schedule-based timing side channel attack are more threatening to static real-time embedded systems where the task characteristics do not change over time.

d: INFLUENCE OF EXECUTION TIME VARIATION

In the methods introduced in Section III, we assume the execution time of each task to be constant. However, in realistic systems the execution time of different jobs have certain variance. In this case, the workload constraint (8) will lead to no feasible solution. To deal with this issue, we can replace the constraint LHS $= L_s$ by constraints LHS $\leq L_s + \epsilon$ and LHS $\geq L_s - \epsilon$ to get better robustness, where ϵ is a constant reflecting the allowed errors. In the following experiments, we set $\epsilon_1 = 10\% \cdot L_s$ and $\epsilon_2 = 20\% \cdot L_s$ based on μ which will be introduced in the following paragraph, and use *pcs-NEW* as the period candidates selection method.

Task set generation: We use normal distribution to simulate execution time variation. The task sets generated in Section IV-B are reused in this experiment to guarantee schedulability and coherency. First, we treat the fixed execution time c_i in the original task sets as worst execution time ($wcet_i$) of each task τ_i in this experiment. Then, for the each release of task τ_i, the specific execution time is sampled from normal distribution $N(\mu, \delta^2)$, where $\mu = wcet_i \cdot 80\%$ and 80% is chosen empirically and the standard deviation δ is calculated with which the cumulative probability $P(X \leq wcet_i)$ is 99.9%. As a result, such a normal distribution produces variation of 95% are within $\pm 10\% \cdot wcet_i$. To guarantee the schedulability of the task set, each specific execution time value is enforced to be WCET if it exceeds WCET.

Performance metric: The purpose of this work is to infer the exact task parameters based on schedule sequences to service for the next step attack behavior as [9] does. One way to evaluate the performance of such algorithm is to compare the task set parameters $\{wcet_i, p_i\}$ with the estimated values $\{\overline{c_i}, \overline{p_i}\}$. Specifically, for each task τ_i in task set Γ, if $\overline{c_i}$ falls into the interval $[wcet_i \cdot 80\%, wcet_i]$ and $[wcet_i \cdot 60\%, wcet_i]$ and $\overline{p_i}$ equals to p_i, we treat it as a successful inference.

Simulation results: To understand the impact that execution time variation brings to the inference precision of our proposed method, we take the simulation results without execution time variation (shown in Figure 11) as the baseline. The results presented in Figure 16, suggest that our proposed method is very sensitive to execution time variation. Specifically, it yields nearly 60% precision ratio decrease when $\epsilon_1 = 10\% \cdot L_s$ and nearly 70% precision ratio decrease when $\epsilon_1 = 20\% \cdot L_s$, in all the task sets. The main reason can be two-fold. i.) The execution time variation will bring diversity to each execution sequence with length of hyper-period. In detail, two adjacent schedule sequence will not be exactly the same with each other, thereby causing more interference in finding the hyper-period value. ii.) As mentioned before, relax of workload constrain will increase the complexity of the MINLP problem. In some worse cases, it is hard for the toolbox (e.g. SCIP, NOMAD) to find a feasible solution.

C. ZEDBOARD-BASED EVALUATION

To test the performance of the proposed method in realistic real-time system, we captured a few execution sequences generated by a Zedboard-based 3DOF helicopter control system. It is important to note that the analysis of execution sequence need not necessarily be achieved on-line due to the deterministic of real-time system. Since our proposed method

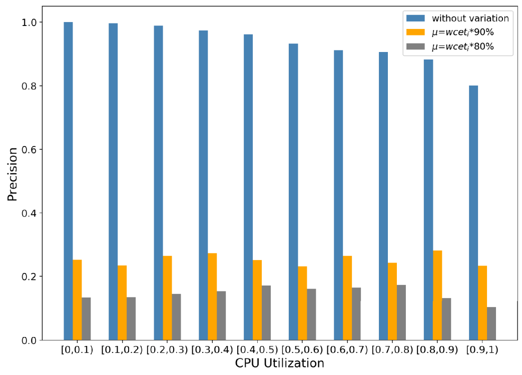


FIGURE 16. Precision comparison of the impact on the precision between with and without execution time variation. The blue bar is the same with which in 11.

uses the optimization toolbox, which is hard to migrate to an on-chip system, in this experiment we only utilize a desktop PC to analyze the captured sequences. In the following, we will introduce the whole experiments in detail.

1) PLATFORM OVERVIEW

The Zedboard [47] (shown in Figure 17) includes a dual-core ARM Cortex-A9 processor and each core has a private 32KB data, 32KB instruction L1 Cache and shares a 512KB L2 Cache. The dual cores also share a Xilinx programmable logic FPGA. The processor runs at 666.67MHz and drives a shared 64-bit global timer (GT) which is clocked at half the CPU frequency. And our control system run on one of the processor core and do not use the FPGA functions. The 3DOF helicopter [48] (displayed in Figure 17) is a simplified helicopter model, equipped with two motors to generate force and three sensors to measure elevation, pitch and travel angle as shown in Figure 17. The Zedboard communicates with the 3DOF helicopter through a PCIe-based control and data acquisition unit [49] and an intermediate Linux-based PC.

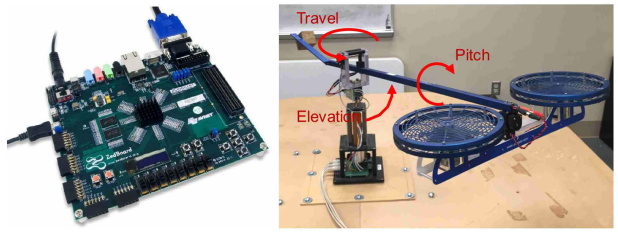


FIGURE 17. Zedboard and 3DOF helicopter.

2) SCHEDULE TRACE CAPTURE

The control system task set is executed on a real-time operating system, FreeRTOS, which is a open source real-time kernel [50]. All tasks are scheduled by fixed priority scheduling algorithm and the detailed information of each task is listed in Table 1. To capture the schedule, the FreeRTOS kernel has been modified on purpose to record the timestamps of each context switch and the preemption of the idle task. More specifically, the idle task is created in the lowest priority with other user tasks by calling *xTaskCreate()* function at system setup. And the idle task can optionally call a *vApplicationIdleHook()* which allows designer to execute their own functions within the idle task. We can write function to recored timestamps referenced by the *Global Timer* when context switch and preemption occur. Both types of time data are saved in a log file and stored in an SD card for the sake of off-line analysis.

TABLE 1. User tasks.

Task Name	Computation Time c_i(ms)	Period p_i(ms)
flight_ctrl	5	20
mission_ctrl	2	100
sensor_rec	2	200
motor_rec	1	400
cam_read	90	600

3) RESULTS

Before we apply our proposed method to the captured traces, the captured busy interval should be refined. Since the busy interval now includes a pair of redundant context switch (idle task in and out). According to our test, this pair of context switch takes about 16.68μs in average on this board. We captured 50 different schedule traces to evaluate the precision ratio of our proposed method, each of them with length 20s. And the execution sequence is obtained by sampling the real schedule trace with 666.67MHz frequency. The performance metric we used is similar to which introduced in Section IV-B2.c. Specifically, for each task τ_i, if the estimated value $\overline{c_i}$ falls into the interval $[c_i - c_i \cdot 20\%, c_i + c_i \cdot 20\%]$ and $\overline{p_i}$ equals to p_i, we treat it as a successful inference. By our method, the task set parameters can be correctly obtained from 8 sequences. And it took around 6.3 hours to finish the overall analysis procedure for each trace.

As the simulation-based and Zedboard-based evaluation show, the execution time variation will bring a huge performance decrease to our method. However, it is important to note that around 20% precision ratio is still useful for adversaries who aim to steal information under detection. In [8] the authors presented that, if the exact task set parameters are given in prior, the *ScheduLeak* algorithm presented in [8] can give at least 93% precision ratio in reconstructing the schedule sequence. With such information, a cache-based side channel attack can be used to precisely measure the cache usage information of an image encode task. Therefore, it is still very promising to perform such attack by combing our method and the *SchedulLeak* algorithm.

V. CONCLUSION

Traditionally, the real-time capability and security are treated as two unrelated aspects in real-time embedded system design. Recently, some work has shown these two aspects actually can be related as the adversary may utilize the timing behavior of the system to infer important information and use them to conduct various attacks. In particular, [8] identifies the schedule-based timing side-channel attack, in which the attacker can reconstruct the exact schedule (telling which task is executed at each time point) of the system from the execution sequence (telling whether there is an task being executed or not, i.e., whether the processor is busy or idle, at each time point).An important assumption in this type of attack is the prior knowledge of attackers about the task parameter information, including the number of tasks in the system and the period and execution time of each task. The attackers can use such information, together with the execution sequence of the task system, to reconstruct the exact schedule of the tasks and perform various subsequent attacks.

In this paper, to make such a schedule-based side-channel attack more practical, we develop effective analysis methods to infer the task parameter information from the execution sequence. Our analysis methods are performed in two steps, first select a set of possible candidates for the task periods, and then compute the execution time of the task corresponding to each period candidate.If the resulting execution time corresponding to a period candidate is 0, it means the system does not contain a task with this period value. We conduct both experiments with synthetic task sets and a case study with a flight control system of civilian drones to evaluate the proposed task parameter analysis method. The experimental results show that the proposed methods work well in most cases. However, it is sensitive to task execution time variations, which needs further improvements in future work. Moreover, we also present and discuss possible ways to obtain the execution sequences.

APPENDIX: DFT BASICS

Discrete Fourier Transform (DFT) [14] is a classical signal processing method that converts a finite sequence of equally spaced samples into the frequency domain. The normalized DFT of a scheduling sequence $x(t), t = 0, 1, \ldots, N$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}}, \quad k = 0, 1, \ldots, N-1 \tag{10}$$

We can also write it in a simple form $X = \mathcal{DFT}[x]$. The absolute value of $X(k)$ indicates the weight of frequency component $\frac{k}{N}$ of $x(t)$. By using inverse Discrete Fourier Transform $x = \mathcal{DFT}^{-1}[X]$, we can return from the frequency domain back to the time domain. Notice that the amplitude spectrum $|X(k)|$ of sequence $x(n)$ is symmetric with respect to frequency $k = \frac{N}{2}$, so it is sufficient to show half of the amplitude spectrum $|X(k)|, k = 1, \ldots, \frac{N}{2}$.

REFERENCES

[1] D. Schneider, "Jeep hacking 101," *IEEE Spectr.*, vol. 8, 2015. [Online]. Available: https://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Secur. Symp. (USENIX Secur.)*, vol. 4, 2011, pp. 447–462.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 447–462.

[4] K. Wesson and T. Humphreys, "Hacking drones," *Sci. Amer.*, vol. 309, no. 5, pp. 54–59, Oct. 2013.

[5] D. P. Shepard, J. Bhatti, and T. E. Humphreys, "Drone hack: Spoofing attack demonstration on a civilian unmanned aerial vehicle," *GPS World*, vol. 23, pp. 30–33, Aug. 2012.

[6] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "S3A: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems," in *Proc. 2nd ACM Int. Conf. High Confidence Netw. Syst. (HiCoNS)*, 2013, pp. 65–74.

[7] R. Pellizzoni, N. Paryab, M.-K. Yoon, S. Bak, S. Mohan, and R. B. Bobba, "A generalized model for preventing information leakage in hard real-time systems," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 271–282.

[8] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 90–102.

[9] C. Y. Chen, A. Ghassami, S. Nagy, M. K. Yoon, S. Mohan, N. Kiyavash, R. B. Bobba, and R. Pellizzoni, "Schedule-based side-channel attack in fixed-priority real-time systems," Tech. Rep., 2015.

[10] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, and L. Sha, "SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 21–32.

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

[12] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Out of control: Stealthy attacks against robotic vehicles protected by control-based techniques," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, Dec. 2019, pp. 660–672.

[13] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2005, pp. 449–460.

[14] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Trans. Audio Electroacoust.*, vol. 15, no. 2, pp. 70–73, Jun. 1967.

[15] S. Vigerske and A. Gleixner, "SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework," *Optim. Methods Softw.*, vol. 33, no. 3, pp. 563–593, May 2018.

[16] S. Le Digabel, "Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm," *ACM Trans. Math. Softw.*, vol. 37, no. 4, pp. 1–15, Feb. 2011.

[17] IBM Analytics. *IBM CPLEX Optimizer*. Accessed: May 19, 2019. [Online]. Available: https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[18] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing real-time microcontroller systems through customized memory view switching," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[19] K.-S. Lhee and S. J. Chapin, "Buffer overflow and format string overflow vulnerabilities," *Softw., Pract. Exp.*, vol. 33, no. 5, pp. 423–460, Apr. 2003.

[20] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 552–561.

[21] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Systematic classification of side-channel attacks: A case study for mobile devices," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 465–488, 1st Quart., 2018.

[22] S. Wakabayashi, S. Maruyama, T. Mori, S. Goto, M. Kinugawa, and Y.-I. Hayashi, "POSTER: Is active electromagnetic side-channel attack practical?" in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 2587–2589.

[23] Y.-I. Hayashi, N. Homma, T. Mizuki, H. Shimada, T. Aoki, H. Sone, L. Sauvage, and J.-L. Danger, "Efficient evaluation of EM radiation associated with information leakage from cryptographic devices," *IEEE Trans. Electromagn. Compat.*, vol. 55, no. 3, pp. 555–563, Jun. 2013.

[24] A. Sayakkara, N.-A. Le-Khac, and M. Scanlon, "Electromagnetic side-channel attacks: Potential for progressing hindered digital forensic analysis," in *Proc. Companion Proc. ISSTA/ECOOP Workshops*, Jul. 2018, pp. 138–143.

[25] A. Sayakkara, N.-A. Le-Khac, and M. Scanlon, "Leveraging electromagnetic side-channel analysis for the investigation of IoT devices," *Digit. Invest.*, vol. 29, pp. S94–S103, Jul. 2019.

[26] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *Proc. Eur. Symp. Res. Comput. Secur.* Berlin, Germany: Springer, 1998, pp. 97–110.

[27] K. Jiang, L. Batina, P. Eles, and Z. Peng, "Robustness analysis of real-time scheduling against differential power analysis attacks," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 450–455.

[28] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's apprentice guide to fault attacks," *Proc. IEEE*, vol. 94, no. 2, pp. 370–382, Feb. 2006.

[29] J. Son and J. Alves-Foss, "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in MLS systems," in *Proc. IEEE Inf. Assurance Workshop*, Jun. 2006, pp. 361–368.

[30] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proc. ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2008, pp. 44–55.

[31] M. Volp, B. Engel, C. Hamann, and H. Hartig, "On confidentiality-preserving real-time locking protocols," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 153–162.

[32] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Mitigating timing side channel in shared schedulers," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1562–1573, Jun. 2016.

[33] N. Tsalis, E. Vasilellis, D. Mentzelioti, and T. Apostolopoulos, "A taxonomy of side channel attacks on critical infrastructures and relevant systems," in *Critical Infrastructure Security and Resilience*. Cham, Switzerland: Springer, 2019, pp. 283–313.

[34] A. Ghassami, X. Gong, and N. Kiyavash, "Capacity limit of queueing timing channel in shared FCFS schedulers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 789–793.

[35] S. Liu, N. Guan, D. Ji, W. Liu, X. Liu, and W. Yi, "Leaking your engine speed by spectrum analysis of real-time scheduling sequences," *J. Syst. Archit.*, vol. 97, pp. 455–466, Aug. 2019.

[36] M.-K. Yoon, S. Mohan, C.-Y. Chen, and L. Sha, "TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–12.

[37] M.-K. Yoon, J.-E. Kim, R. Bradford, and Z. Shao, "TaskShuffler++: Real-time schedule randomization for reducing worst-case vulnerability to timing inference attacks," 2019, *arXiv:1911.07726*. [Online]. Available: http://arxiv.org/abs/1911.07726

[38] K. Krüger, M. Volp, and G. Fohler, "Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems," *LIPIcs-Leibniz Int. Proc. Informat.*, vol. 106, p. 22, 2018.

[39] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 103–116.

[40] O. Iegorov, R. Torres, and S. Fischmeister, "Periodic task mining in embedded system traces," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2017, pp. 331–340.

[41] S. Mohan, M. K. Yoon, R. Pellizzoni, and R. Bobba, "Real-time systems security through scheduler constraints," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 129–140.

[42] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *Proc. ACM/IEEE 9th Int. Conf. Cyber-Physical Syst. (ICCPS)*, Apr. 2018, pp. 10–21.

[43] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Preserving physical safety under cyber attacks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6285–6300, Aug. 2019.

[44] Agilent. (2014). *N9030A Real-Time Spectrum Analyzer User's and Programmer's Reference*. Accessed: May 19, 2019. [Online]. Available: http://cp.literature.agilent.com/litweb/pdf/N9030-90059.pdf/

[45] Texas Instruments. (2014). *MSP430 Ultra-Low-Power Microcontrollers*. Accessed: May 19, 2019. [Online]. Available: http://www.ti.com/tool/MSP-EXP430G2

[46] ROHDE&SCHWARZ. *R&S HZ-15 Compact Probe Set for E and H Near-Field Measurements, 30 MHz to 3 GHz*. Accessed: May 19, 2019. [Online]. Available: https://www.rohde-schwarz.com/us/product/hz15-productstartpage_63493-8985.html

[47] Xillinx. *Xillinx Zedboard*. Accessed: May 19, 2019. [Online]. Available: http://zedboard.org/

[48] Quanser. *Quanser 3-DOF Helicopter*. Accessed: May 19, 2019. [Online]. Available: https://www.quanser.com/products/3-dof-helicopter/

[49] Quanser. *Q8-USB Data Acquisition Device*. Accessed: May 19, 2019. [Online]. Available: http://www.quanser.com/Products/q8/

[50] FreeRTOS. *The FreeRTOS Kernel: Market Leading, De-Facto Standard and Cross Platform RTOS Kernel*. Accessed: May 19, 2019. [Online]. Available: https://www.freertos.org/