A New Covert Channel in Fixed-Priority Real-Time Multiframe Tasks

Mohammad Fakhruddin Babar and Monowar Hasan
School of Electrical Engineering & Computer Science, Washington State University, USA
Email: m.babar@wsu.edu, monowar.hasan@wsu.edu

***Abstract*—This study investigates the presence of illicit information flows in fixed-priority multiframe real-time systems. We identify an algorithmic covert channel (called FrameLeaker) that enables a low-priority task to deduce the execution patterns (frames) of a high-priority task. We show that the response time of a targeted low-priority task (receiver) can be used to extract the execution behavior of a high-priority task (sender). We further introduce a metric called "inference precision ratio" to evaluate the efficacy of the received information.**

***Index Terms*—Covert Channel, Real-time System, Multiframe.**

I. Introduction

In high-assurance systems such as those with real-time requirements, information flows must be *explicit* (i.e., passed through authorized channels.). Any communication between real-time tasks must be known and configured a priori. As third-party vendors often supply various applications in modern real-time systems, it is challenging to thoroughly trust or verify them, leading to new vulnerabilities. A covert timing channel refers to an unauthorized communication path where one entity (sender) transmits information to another entity (receiver) in violation of security policies. In safety-critical real-time systems (e.g., avionics, automotive, industrial control systems), it is of utmost importance to provide strong isolation among applications that require different levels of criticality to limit interference among them. Hence, there must not exist any illicit information flow (viz., covert channels) between tasks. We identify that the deterministic nature of real-time schedulers can contribute to one such channel in multiframe fixed-priority systems.

Multiframe real-time tasks [1] are used in many industrial applications where execution time is different from one phase to another of its execution. For instance, a periodic task in a control system may collect a small amount of data collection in each period that takes a small execution time but then summarize and store this data every k cycle using a sophisticated algorithm that takes a larger execution time. Another example is in real-time multimedia encoding, where different encodings may take a variable time.

This research is partly supported by the US National Science Foundation (Award 2312006). Any findings, opinions, recommendations, or conclusions expressed in the paper are those of the authors and do not necessarily reflect the sponsor's views.

979-8-3503-7128-4/24/$31.00 ©2024 IEEE

The research aims to formally analyze the creation and investigation of a covert timing channel by a high-priority sender task and a low-priority receiver task in the presence of a third-party task under the rate-monotonic scheduling. We show that by analyzing the response time of the receiver, it is possible to signal the execution behavior of the high-priority sender. By exploiting this channel, a rogue task can *signal its frame patterns* to another task. We name this covert channel vulnerability FrameLeaker. Knowing such information can leak critical information to other parties. For instance, in the control task example, frame patterns can leak whether a task is performing data collection or logging/processing. Our analysis shows an illegitimate communication path between a high and low-priority task, even with the presence of other benign tasks. We formally show that the receiver task can predict at least one of the frames from the sender from its runtime delay (viz., response time). We further introduce a metric (called *inference precision ratio*) that quantity the goodness of received information.

In this paper, we made the following contributions.

- We propose an analytical model that finds the *existence of an algorithmic covert channel* (named FrameLeaker) for a set of multiframe tasks running using the rate monotonic scheduling policy (Section III);
- We formally characterize the channel and prove that it will *always* be possible for a low-priority receiver task to infer at least one frame of the sender task (Section III); and
- We devise a *metric* (called inference precision ratio) to determine the chances of successful inference (Section IV).

II. Background and Models

A. Preliminaries: Covert Channels

A covert channel is a hidden communication channel. Such channels are not intended to exist in the system and are created furtively by untrusted entities using techniques antithetical to the system's intended design. From a real-time perspective, we investigate how malicious tasks can exploit deterministic scheduling policies to illicitly transfer information (frame patterns in our context).

A covert channel can be classified into two major categories: *(a)* noiseless channel and *(b)* noisy channel. A covert communication path can be represented as a *discrete*

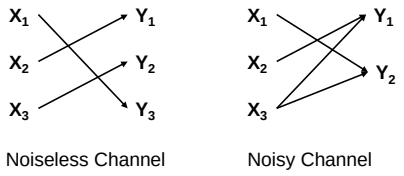


Fig. 1. Types of covert channels: *(a)* output alphabets directly map a particular input alphabet (noiseless channel, left); and *(b)* multiple input alphabets can result in the same output alphabets (noisy channel, right).

memoryless channel, where information flows from a *sender* to a *receiver*, as illustrated in Figure 1. The channel is *discrete* when the input alphabets $X = \{x_1, \ldots, x_j\}$ and the output alphabets $Y = \{y_1, \ldots, y_j\}$ are finite. It is *memoryless* when the current output depends only on the current input. When noise occurs during the transmission, it affects the output the receiver observes. The behavior of a noisy channel can be nondeterministic, meaning that the output observed by the receiver is no longer solely determined by the input symbol transmitted. For instance, in the noisy channel depicted in Fig. 1, upon receiving y_1, the receiver cannot reliably determine which value (x_2 or x_3) was the input transmitted by the sender.

B. System Model

We consider a uniprocessor real-time system with three periodic multi-frame tasks i.e., τ_H, τ_N, and τ_L. A task τ_H intends to secretly leak its execution behavior to a τ_L in the presence of a high-priority task τ_N. Hence, τ_N acts as a "noise" in the covert channel between τ_H and τ_L. We donate $hp(\tau_i)$ as the set of high-priority tasks than τ_i. By assumption, $hp(\tau_L) = \{\tau_N, \tau_H\}$ and τ_N is the highest-priority task. We further assume that the tasks are scheduled using rate monotonic [2] scheduling policy.

Each task τ_i is characterized by $\tau_i = \{\mathcal{C}_i, T_i, D_i, n_i\}$, where $\mathcal{C}_i$ is the set of worst-case execution times of the task, T_i is the inter-arrival time (period), D_i is the relative deadline, and n_i denotes the repeating sequence of *frames* [1]. If a task has $n_i \geq 1$ frames, then $\mathcal{C}_i = \{C_i^0, C_i^1, \cdots, C_i^{n_i-1}\}$. Each frame may have a worst-case execution time that may differ from other frames. All frames in the same task are released with the time interval T_i and have the same deadline D_i.

TABLE I
MULTIFRAME TASKSET

Task	T	C	Priority
τ_1	10	{1,3,5}	High
τ_2	20	2	Low

Table I shows an example with 2 tasks τ_1 and τ_2 where τ_1 is a multi-frame task with 3 frames represented by the execution time values 1, 3, and 5; and τ_2 has just one frame. We further assume the tasks follow the implicit deadline model ($D_i = T_i$), i.e., the tasks must complete before their next periodic arrival.

C. Threat Model

We consider a noninterference model [3], i.e., the scheduling parameters, the scheduling algorithm, and the resulting schedule are publicly available to the adversary. Besides, all tasks have access to precise clocks. We aim to analyze whether a rouge task τ_H can form a covert channel by exploiting the deterministic nature of real-time schedules. Particularly, the high-priority task (τ_H) wants to *leak* its execution patterns (frames) to a low-priority one (τ_H) in the presence of another benign task (τ_N).

III. FRAMELEAKER: ALGORITHMIC COVERT CHANNELS IN MULTIFRAME REAL-TIME SYSTEM

We demonstrate a vulnerability in multiframe scheduling where a low-priority task can infer another high-priority task's varying execution behavior by observing the impact that the low-priority task has on its own response time (i.e., the time between arrival and completion of a frame). We refer to this FrameLeaker. For instance, if τ_H consists of n_H frames, by analyzing the response time of τ_L, we infer what frames τ_H is transmitting at a given point in time. In this case, the two parties do not require any explicit communication path. The receiver task (τ_L) can extract the sending signal by observing its completion time (say, by reading the system clock). Hence, we refer to this illegitimate information flow as *algorithimic covert channel*.

As an example, consider a taskset presented in Table II, where τ_H has three frames (i.e., $C = \{1, 2, 3\}$). Depending on the τ_H's frame execution, the response time of τ_L varies. To illustrate, when the active frame is 3, the response time of τ_L is 9, and when the active frame is 2, the response time of τ_L is 8 (see Fig. 4). We represent the FrameLeaker covert channel vulnerability as a communication model described as a set with input-output binary relations. We do so by extending the response time analysis for multiframe task models considering *all possible invocations* of the receiver task, as we present below.

TABLE II
EXAMPLE TASKSET 1

Task	T	C
τ_N	5	2
τ_H	10	{1,2,3}
τ_L	20	2

A. Response Time Analysis for Multiframe Tasks

The response time R_i of a task τ_i consists of two types of execution: the task's execution and the delay caused by the execution of other higher-priority tasks (called *interference*). Let $\pi(\tau_i)$ represent a scheduling priority assigned to a task τ_i. A taskset Γ running a fixed-priority scheduling policy will meet deadlines for all tasks if [4]: $\forall \tau_i \in \Gamma\ R_i \leq D_i$, where $R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \lceil \frac{R_i}{T_j} \rceil C_j$.

Assume a multiframe task τ_j with n_j execution frames $\{C_j^0, C_j^1, \cdots, C_j^{n_j-1}\}$. Let us define a cumulative function

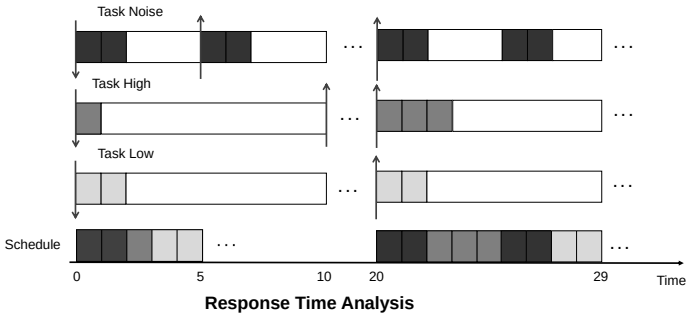


Fig. 2. Response time analysis of Task τ_L for Taskset 1 (see Table II). τ_L releases second instance at $t = 20$. Frame sequence of task τ_H is $\{3, 1, 2\}$. We can see that $R_L(1) = 9$ and high is transmitting frames where $C_H = 3$.

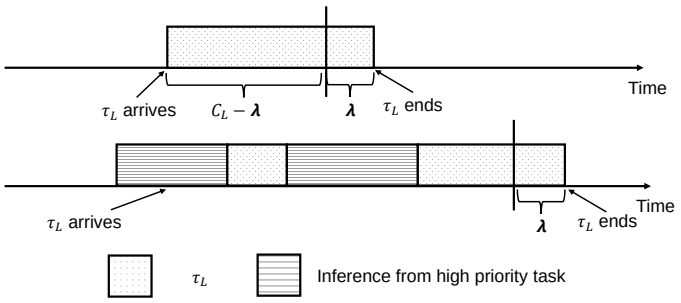


Fig. 3. Execution of τ_L has $C_L - \lambda$ unit of time for computation and λ time for response time calculation. The bottom part depicts high-priority tasks (τ_N and τ_L) preempt τ_L.

$\xi_j(k)$ that calculates the summation of the execution time of different frames of τ_j up to a certain invocation k and is defined as follows [1]:

$$\xi_j(k) = \sum_{i=1}^{i=k} C_j^{i \bmod n_j}. \tag{1}$$

To illustrate, the value $\xi_j(3)$ for task τ_j whose execution times are $\{5, 8, 3, 4, 2\}$ is 16. The value $\xi_j(3)$ for task τ_j whose execution times are $\{5, 8, 2, 4, 2\}$ is 15. For a single-frame task, the cumulative function is $\xi_j(k) = kC_j$ because of the constancy of C_j for all frames of the multiframe task.

Based on this function $\xi_j(k)$, the response time for the multiframe task model is represented as [1]: $R_i = C_i^m + \sum_{\tau_j \in hp(\tau_i)} \xi_j(\lceil \frac{R_i}{T_j} \rceil)$, where $C_i^m \in \mathcal{C}_i$ is called the "peak frame" that leads to worst-case behavior. We can solve the above using the following standard recurrence process:

$$R_i^{w+1} = C_i + \sum_{\tau_j \in hp(\tau_i)} \xi_j(\lceil \frac{R_i^w}{T_j} \rceil), \tag{2}$$

where $w = 0, 1, 2, 3, \cdots$ and $R_i^0 = C_i$. The *worst-case* response time is obtained when $R_i^{w+1} = R_i^w$ for the smallest value of w.

B. Extended Timing Analysis for Multiframe Tasks

To expose FrameLeaker covert channel vulnerability (frame length) from τ_H, we extend Eq. (2) to calculate all possible response times of τ_L. Before we present details of our analysis, let us start with a definition.

Definition 1 (Active Frame)**.** *The frames in a multiframe task arrive sequentially at regular intervals. The frame ready to be executed at a given time* t *is defined as the* ***active frame****. We denote the active frame of the* k^{th} *job of task* τ_i *as* $C_i(k)$.

For example, if task τ_i has three frames $\{2, 3, 5\}$, for the second job of τ_i, the active frame is 3. Likewise, the active frame for the fourth job of τ_i is 2.

We measure the response time of jobs arriving at T_L interval to observe which frames of τ_H transmit during that time. Hence, we calculate response time of k^{th} released task of τ_L as follows:

$$R_L^{w+1}(k) = C_i(k) + \sum_{\tau_j \in \{\tau_N, \tau_H\}} \xi_j(\lceil \frac{R_i^w(k)}{T_j} \rceil) \tag{3}$$

Using Eq. (3), we deduce the response time of τ_L at different instance, i.e., $R_i(k)$, where $k = 1, 2, 3, \cdots$. For the first instance of task τ_L (i.e., first job), the response time is $R_L(1)$. Additionally, by assumption $R_L \leq D_L$ (i.e., the task meets the deadline), and hence *schedulable*.

C. Identifying Frame Signals

By leveraging the noninterference model, the receiver task keeps track of time read from the system time, including preemptions. For correct system operation (and remain stealthy) τ_L should not run more than its worst-case execution time, C_L. Hence, it saves some budget for analyses to reconstruct (passive) information received from the sender. Hence, we define a parameter, λ, whose value is set by the receiver to limit the running time of the inference function in each period. This inference duration λ, is an integer in the range $0 < \lambda < C_L$, as shown in Fig. 3.

We use the active frames to deduce the signals from τ_H. Hence, we calculate the response time of τ_L when both τ_H and τ_L arrive simultaneously, i.e., at multiple of the least common multiple (LCM) of T_H and T_L. We then determine the response time of τ_L at those points. Considering the response time of τ_L at any other time may introduce interference from the previously released frame and the effect of the upcoming frame (i.e., create more noise in the inference). In contrast, the response time at the multiple of LCM of T_H, and T_L is *noiseless* as both τ_H and τ_L arrive simultaneously, and τ_L can deduce the frame that τ_H is transmitting correctly.

We use the following indexing to find the initial position of a multiframe of τ_H at different times. If τ_H has n_H frames, then for k^{th} release of the τ_L, frame initial index p will be calculated as follows:

$$p = \lceil \frac{t}{T_L} \rceil \bmod n_H,\ t = k \times T_{LCM},\ k = 0, 1, 2, 3, \cdots, \tag{4}$$

Algorithm 1 FrameLeaker Inference: Inferring frames of high-priority task τ_H

1: **Input:** $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$
2: **Output:** Deduced frame of τ_H by τ_L
3: *Length of multi-frame of sender task* $\leftarrow n_H$
4: $T_{LCM} \leftarrow LCM(T_L, T_H)$
5: $t\_max \leftarrow n_H \times T_{LCM}$
6: $t \leftarrow 0$
7: $\mathcal{C}\_deduce \leftarrow \{\}$ ▷ *Store the deduced frame of* τ_H
8: $R_L \leftarrow \{\}$
9: **while** $t < t_{max}$ **do** ▷ *Each job of* τ_L *will calculate response time* R_L *till* t_{max}
10: $index = \lceil \frac{t}{T_H} \rceil \bmod n_H$ ▷ *index flag is used to detect the active frame of* τ_H
11: $R_L \leftarrow R_L(t/T_L)$ ▷ *Calculate* $R_L(t/T_L)$ *using Eq.* (3)
12: $\mathcal{C}\_deduce \leftarrow \mathcal{C}_H[index]$
13: $t \leftarrow t + T_{LCM}$ ▷ *Response times at multiple of* T_{LCM}
14: **end while**
15: `set`($\mathcal{C}\_deduce$) ▷ *Getting the deduced frame comparing* R_L *&* $\mathcal{C}\_deduce$. `set`() *returns distinct elements*

where T_{LCM} is $LCM(T_L, T_H)$. To illustrate, let us assume τ_H has three execution frames (e.g., $\{1, 2, 3\}$) and for 2nd release of τ_L, frame sequence of τ_H will be $3, 1, 2, \cdots$ and so on.

D. Algorithm

FrameLeaker's inference process is presented in Algorithm 1. The variable T_{LCM} stores the LCM of T_L and T_H (Line 4). We calculate the response time of τ_L at a multiple of T_{LCM} until a timeout period, t_{max} (Line 5). We store the deduced frame of τ_H in $\mathcal{C}\_deduce$ (Line 7) and response time of τ_L in R_L (Line 8). We find the active frame index of τ_H (Line 10) and calculate R_L using Eq. (3) (Line 11). We store the deduced frame of τ_H in $\mathcal{C}\_deduce$ (Line 12) and infer the deduced frame comparing R_L and $\mathcal{C}\_deduce$ (Line 15).

E. Examples and Illustrations

We illustrate the inference process with a few examples.

Example 1. Assume we have three tasks τ_N, τ_H, τ_L with periods 5, 10, and 20. τ_H has three frames 1, 3, and 5, as listed in Table II. Response time of τ_L varies because of the multiframe nature of the τ_H. We infer which frame τ_H is transmitting by analyzing the response time of τ_L. To evaluate the response time of τ_L, we need to find the frame sequence of τ_H at the time of k^{th} arrival of τ_L. We can see the frame sequence of τ_H at $t = 0, 20, 40$ in Fig. 4. Using Eq (3), we calculate the response time of the first instances of τ_L. $R_L(k)$ denotes the response time of τ_L at k^{th} release. Using Eq. (3), $R_L(1) = 5$, $R_L(2) = 9$, $R_L(3) = 8$.

Example 2. Assume we have three tasks τ_N, τ_H, τ_L with periods 4, 8, and 12. τ_H has three frames with durations $\{1, 2, 3\}$ as listed in Table III. τ_L calculates the frame sequence of τ_H at the time of k^{th} arrival of τ_L. In this example, the frame sequence of τ_H at $t = 0, 24, 48$ (multiple of LCM of T_L, T_H, see Fig. 5). Using Eq. (3), $R_L(1) = 5$. In this case, τ_L can only infer τ_H's frame duration when τ_H transmits its first

τ_H Frame Seq.	$R_L(k)$
(1,2,3)	$R_L(1)$ = 5
(3,1,2)	$R_L(2)$ = 9
(2,3,1)	$R_L(3)$ = 8

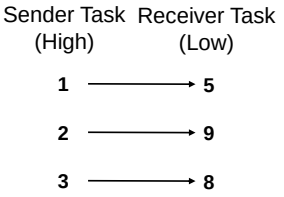


Fig. 4. Response times and input/output signal mapping for Taskset 1.

τ_H Frame Seq.	$R_L(k)$
(1,2,3)	$R_L(1)$ = 5

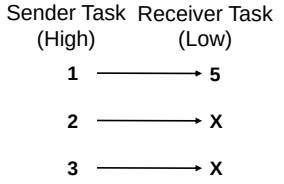


Fig. 5. Response times and input/output signal mapping for Taskset 2.

TABLE III
EXAMPLE TASKSET 2

Task	T	C
τ_N	4	1
τ_H	8	{1,2,3}
τ_L	12	2

frame. Due to the parameters of the system (i.e., period and number of frames), we cannot extract more than one frame. For instance, in this taskset $\frac{\text{LCM}(T_H, T_L)}{T_H} = n_H$. As a result, at each checking point (i.e., multiple of $LCM(T_L, T_H)$), τ_H transmits the same frame. Thus, τ_L cannot infer more than one frame.

F. Characterizing the Channel

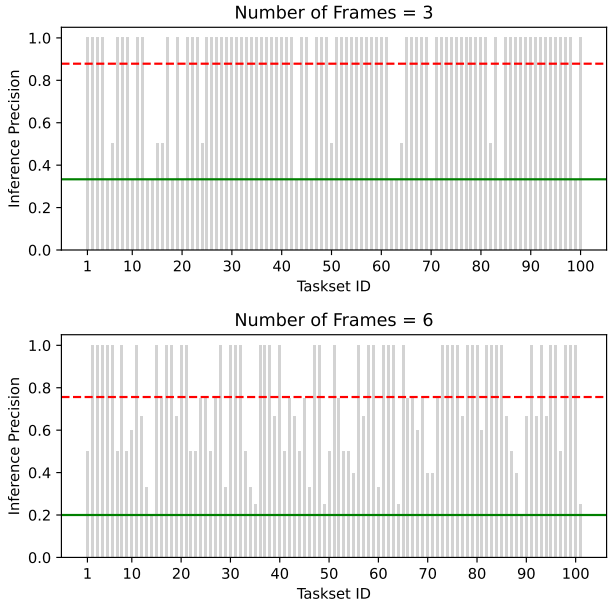
We now characterize the FrameLeaker covert channel. Let us introduce two definitions.

Definition 2 (Channel Deducibility)**.** *A channel is* ***fully deducible*** *if task* τ_L *can infer all the frames of task* τ_H*. Likewise, a channel is* ***partially deducible*** *if task* τ_L *can infer some, but not all, frames of task* τ_H*.*

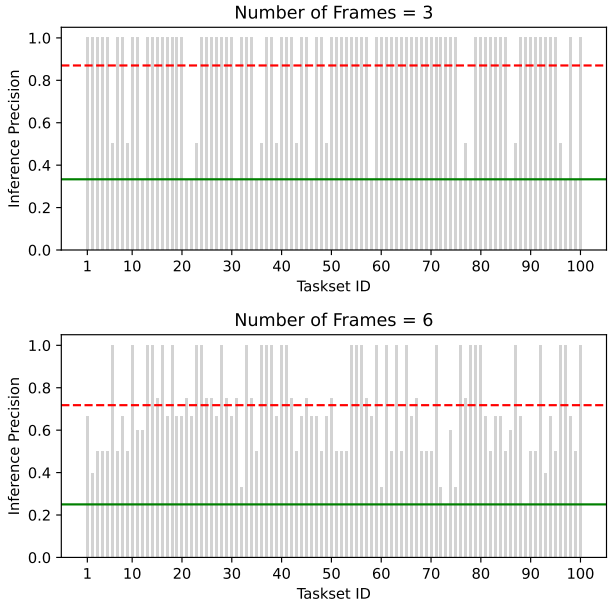
By observing its response time, τ_L can infer which frame is transmitted by the task τ_H. For instance, as in Example 1, τ_L can map the frame sequence τ_H and the corresponding response time (Fig 4). As we shall present below, it is always possible to extract information about at least one frame, and hence, the covert channel we discovered is *partially deducible*.

Lemma 1. *The noisy channel in FrameLeaker is always at least partially deducible for a schedulable taskset.*

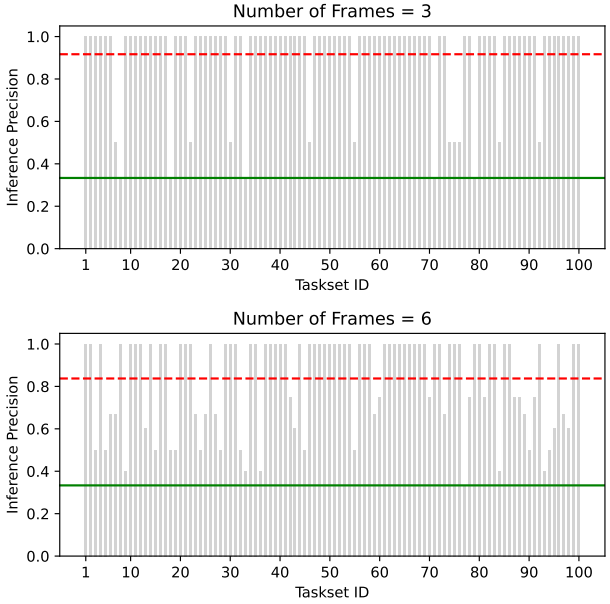
Proof. We deduce the frame of sender task τ_H using the response time of τ_L at a multiple of LCM of T_L and T_H, i.e., $T_{LCM} = LCM(T_L, T_H)$. We check response time at $t = k \times T_{LCM},\ k = 0, 1, 2, 3, \cdots$, when both τ_L, τ_H arrive simultaneously. In these values of t, the response time of τ_L



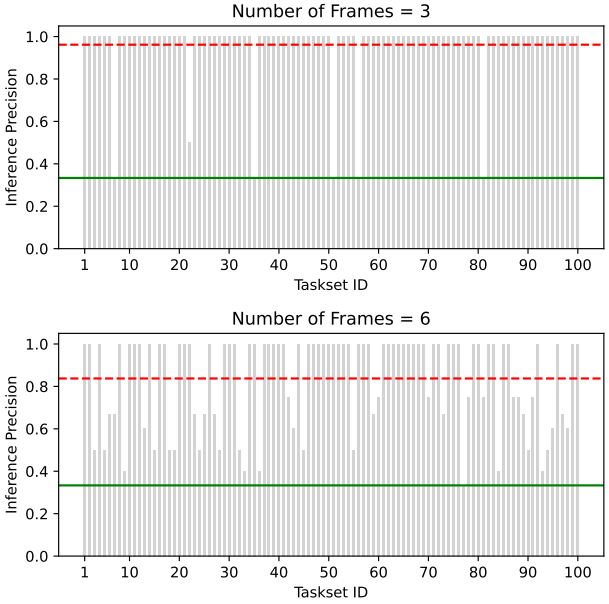
(a) Inference precision: periods of τ_H and τ_L generated randomly.



(b) Inference precision: periods of τ_H and τ_L are harmonic.



(c) Inference precision: fixed the period of τ_L with varying period of τ_H.



(d) Inference precision: fixed the period of τ_H while τ_L's period is varied.

Fig. 6. Inference precision for 100 randomly generated tasks with fixed frame length (3 and 6) for four different cases: *(a)* T_L and T_H are randomly generated, *(b)* T_L and T_H are harmonic, *(c)* Fixed T_L and varied T_H, and *(d)* Fixed T_H and varied T_L. The red line indicates the average precision value, while the green line indicates the minimum precision value.

has no interference from the previous frame of τ_H. At each multiple of T_{TCM}, τ_L may receive different frames of τ_H. But in the worst case scenario, the frame of τ_H comes in such a way that where at each multiple of T_{LCM}, exactly the same frame of τ_H arrives. This happens when $\frac{T_L}{T_H} = n_H$. Thus, we can conclude that even in the worst-case scenario, τ_L can deduce at least one frame of τ_H. □

IV. Evaluation

We measure the information leakage in FrameLeaker using synthetically generated workloads. We evaluated using 100 randomly generated task sets under the following setups: *(a)* T_L and T_H are randomly generated, *(b)* T_L and T_H are harmonic, *(c)* Fixed T_L and varied T_H, and *(d)* Fixed T_H and varied T_L. In randomly generated case, the task periods are

TABLE IV
SIMULATION PARAMETERS

Parameters	Value
Period T_H	$[10, 15]$ ms
Period T_L	$[30, 50]$ ms
Number of frames, n_i	$\{3, 6\}$
Number of taskset for each utilization, N_u	100

TABLE V
AVERAGE AND MINIMUM PRECISION

Case	No. of Frames	Avg. Precision	Min. Precision
Random	3	0.88	0.33
Random	6	0.75	0.20
Harmonic	3	0.87	0.33
Harmonic	6	0.72	0.25
Fixed T_L	3	0.92	0.33
Fixed T_L	6	0.84	0.33
Fixed T_H	3	0.96	0.33
Fixed T_H	6	0.83	0.25

randomly selected from $[15, 30]$ ms for τ_H and $[50, 70]$ ms for τ_L. Table IV lists the key simulation parameters. In the harmonic period case, we vary the ratio of $\frac{T_L}{T_H}$ in the range of $[2, 5]$. In the fixed T_L case, we set $T_L = 120$ and vary T_H in the range of $\frac{T_L}{[2,5]}$ (i.e., $T_H < T_L$). In the fixed T_H case, we set $T_H = 30$.

Metric: Inference Precision Ratio. We propose a *metric* to measure the successful inference of receiving frames. This metric represents the ratio of detectable frames to the total frame length and is calculated as follows:

$$Q = \frac{\text{Total number of deducible frames } (n_d)}{\text{Total number of frames } (n_j)}$$

For instance, in Example 1, $Q = 3/3 = 1$, while in Example 2, $Q = 1/3 = 0.33$. These relative quantities can serve as effective metrics for comparing the deducibility levels of two noisy channels. For instance, when the Q of noisy channel A for a given task set exceeds that of channel B for another one, we may conclude that the receiver can deduce more information about the frame sequence.

A. Results

We include the average inference precision (red line) and minimum precision (green line) in each plot (see Fig. 6). The average precision and minimum precision values for all four cases are listed in Table V. Inference is better for lower frame numbers (see Figs. 6). When we fix T_H and vary T_L, the best average precision value is achieved (i.e., 0.96). Based on the minimum precision value and also from Lemma 1, we can conclude it is possible to deduce at least one frame in the worst-case scenario (i.e., inference precision value is always greater than 0). In all of our experimental cases, we observed that the 90^{th} percentile value is 1, indicating that our framework achieves very high inference precision.

V. RELATED WORK AND CONCLUSION

While some work exists on securing real-time systems [5]–[7], there has not been much work on exploring attack surfaces. Perhaps the closed work is Son et al. [8] that identified timing covert channels for rate monotonic schedulers. However, this early work does not provide any analytical verification. There exists other work [3], [9]–[13] for discovering side and covert channel leakages, but none of them have considered multi-frame scenarios.

FrameLeaker exposes an algorithmic covert channel in multi-frame task scenarios. Under the assumption that τ_L is a task with the longest period, we show that the low-priority task least infers the transmission of one frame of a multi-frame task while in the best case, we can infer all of them! Our findings will help designers to be cognizant of covert channel leaks and modify schedulers to prevent such information flows.

REFERENCES

[1] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.

[2] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, pp. 166–171, 1989.

[3] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pp. 44–55, 2008.

[4] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[5] 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 *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 1–12, IEEE, 2016.

[6] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Integrating security constraints into fixed priority real-time schedulers," *Real-Time Systems*, vol. 52, pp. 644–674, 2016.

[7] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba, "Contego: An Adaptive Framework for Integrating Security Tasks in Real-Time Systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, pp. 23:1–23:22, 2017.

[8] J. Son *et al.*, "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems," in *2006 IEEE Information Assurance Workshop*, pp. 361–368, IEEE, 2006.

[9] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 90–102, IEEE, 2019.

[10] M. Völp, B. Engel, C.-J. Hamann, and H. Härtig, "On confidentiality-preserving real-time locking protocols," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 153–162, IEEE, 2013.

[11] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Mitigating timing side channel in shared schedulers," *IEEE/ACM transactions on networking*, vol. 24, no. 3, pp. 1562–1573, 2015.

[12] X. Gong and N. Kiyavash, "Quantifying the information leakage in timing side channels in deterministic work-conserving schedulers," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1841–1852, 2015.

[13] J. Kwak and J. Lee, "Covert timing channel design for uniprocessor real-time systems," in *Parallel and Distributed Computing, Applications and Technologies: 19th International Conference, PDCAT 2018, Jeju Island, South Korea, August 20-22, 2018, Revised Selected Papers 19*, pp. 159–168, Springer, 2019.