Project Report
TaskShuffler: A Schedule Randomization Protocol for Obfuscation Against Timing Inference Attacks in Real-Time Systems

Ankita Samaddar

I. Introduction

Due to increased computational power and connectivity in real-time systems, threats in the modern real-time systems are increasing day by day. Some of these real-time systems are critical in nature. A successful attack in these systems may lead to disastrous effects ranging from loss of human lives to damages to machines. Therefore, there is a need to provide measures to protect these systems from attackers. Each real-time task in a real-time system is associated with a deadline within which the task has to be completed. This timeliness guarantee of each real-time task in a real-time system enforces the schedule of these task systems to be predictable. Hence, designers need to ensure that these systems have very narrow range of operations and fixed modes of execution to avoid safety breaches. Any perturbations to the operational modes can lead the entire system to the unsafe state. On the other hand, adversaries can attack a system by analyzing the predictable patterns of execution in real-time systems. The most common example of such an attack is the Side Channel Attack (SCA), where an attacker can observe the execution pattern of real-time tasks and get to know about the execution pattern from the predictable schedule. From this predictable task schedule, the attacker may guess the execution pattern and launch a SCA on the intended task. These attacks can succeed due to the limited uncertainty in the schedule in the repeating schedules generated by the periodic real-time tasks. If an attacker gets the schedule over an hyperperiod, he can predict the the future schedules very precisely.

As a part of my RTOS project, I implemented the a schedule randomization protocol, knwn as the TaskShuffler, for schedule obfuscation against timing inference attacks in real-time systems [3]. In [3], a schedule randomization protocol called the TaskShuffler has been proposed to reduce the determinism in the schedule by the adversaries in real-time system. The TaskShuffler generates random schedules which are highly unpredictable but still can meet the real-time guarantees of the system. TaskShuffler selects a random job from the Ready Queue irrespective of its priority based on some constraints that still enforces all the tasks to meet their deadline. Hence, adversaries are unable to deduce the exact order of execution of the task system. Though the adversary has prior knowledge of the system, yet it becomes very difficult to initiate a timing inference attack on a random schedule.

II. System Model

We consider a uniprocessor system with a set of n periodic tasks, represented as $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is denoted as (e_i,p_i,d_i), where e_i is the worst-case execution time of i^{th} task, p_i is the inter-arrival time or period between successive releases of tasks, d_i is the relative deadline. We have assumed that our tasks have implicit deadlines, i.e., $d_i = p_i$. Therefore, only one job per task instance is available at a particular time instant. The tasks are assigned priorities based on Rate Monotonic (RM) algorithm. Priority of i^{th} task is denoted as $Pri(\tau_i)$. $hp(\tau_i)$ denotes the set of all tasks which have higher priority than τ_i. Similarly, $lp(\tau_i)$ denotes the set of all tasks having priority lower than τ_i. We have considered that the tasks set τ is schedulable by fixed-priority preemptive scheduling. Hence, the worst-case execution time of task i, $wcrt_i$ is less than or equal to deadline d_i, i.e., $wcrt_i \leq d_i$. The $wcrt_i$ is calculated by iterative response time analysis, [1],given by Eq. (1), where, $r_i^0 = e_i$, $wcrt_i = r_i^{k+1} = r_i^k$ for some k, e_i,p_i and $d_i \in \mathbf{N}^+$.

$$wcrt_i = e_i + \sum_{\tau_j \in hp(\tau_i)} \lceil \frac{r_i^k}{p_j} \rceil e_j \leq d_i \tag{1}$$

III. Adversary Model

We assume that our adversary has the information about the timing parameters of the real-time tasks. If an adversary has the knowledge of the timing parameters of the tasks, he can easily launch a side channel attack (SCA) or a covert channel attack on the system. Both of these attacks succeed if the schedule is deterministic in nature, which is the case in real-time systems due to the constraint of deadline satisfaction of real-time tasks. Our objective is to randomize the schedule and make the schedule unpredictable by the adversary. We assume that our scheduler is trustworthy.

IV. TaskShuffler: Schedule Randomization Protocol

This section discusses the TaskShuffler protocol which randomizes the schedule to reduce the inferability of the schedule

for a real-time task system. Due to randomization of schedule, the execution order of the real-time tasks vary from one hyper-period to another. Hence, even if an observer is able to record the exact schedule for a hyper-period, there is no guarantee that the same execution pattern will show up in the next hyper-period thereby enforcing the schedule to be less predictable. To apply the TaskShuffler, we need to calculate two other parameters of every real-time tasks-

1) Worst-Case Maximum Inversion Budget (V_i)- The worst-case maximum inversion budget of task τ_i is the maximum amount of time for which all lower priority tasks $lp(\tau_i)$ are allowed to execute while an instance of τ_i is still active (i.e., in the ready queue) while meeting its deadline even in the worst-case scenario. It is calculated by-

$$V_i = d_i - (I_i + e_i) \tag{2}$$

where d_i is the deadline of τ_i, e_i is the execution time of τ_i and I_i is the maximum interference of tau_i from its higher priority tasks, $hp(\tau_i)$. I_i is given by-

$$I_i = \sum_{\tau_j \in hp(\tau_i)} (\lceil \frac{d_i}{p_j} \rceil + 1) e_j \tag{3}$$

2) Minimum inversion priority (M_i)- The minimum inversion priority of a task τ_i, is the minimum priority that can delay τ_i. It is defined by the highest priority among $lp(\tau_i)$ that has a negative worst-case maximum inversion budget. M_i is calculated as-

$$M_i = max(Pri(\tau_j) | \tau_j \in lp(\tau_i) and V_j < 0) \tag{4}$$

When there is no such a task, then M_i is set to an arbitrarily minimum priority.

These two parameters are calculated offline for every real-time tasks in the taskset. The pseudocode for the TaskShuffler is given in Algorithm 1.

Algorithm 1 TaskShuffler $(t, \tau, L_{\mathcal{R}})$

t: the current time instant
τ: the task set τ
$L_{\mathcal{R}}$: the Ready Queue sorted in descending order of priority. $L_{\mathcal{R}} = \{\tau_1, \tau_2, \ldots, \tau_{|L_{\mathcal{R}}|}\}$

```
L_c ← τ_1   ▷ the highest priority task in the Ready Queue
if v_(1) < 0 then
    return τ_1
end if
for τ_i = τ_2, ..., τ_(|L_R|) do
    if Pri(τ_i) ≥ M_(1) then
        L_c ← L_c ∪ τ_i
    end if
    if v_(i) ≤ 0 then
        Stop the iteration
    end if
end for
id ← random(1, |L_c|)   ▷ random number selection between 1 and |L_c|
τ_s ← L_c[id]   ▷ select the task with the available id
if id > 1 then
    δt = [1, min(v_j | τ_j ∈ hp(τ_s) ∩ R)]
    schedule next decision at t + δt
end if
```

The TaskShuffler maintains two queues- a)a Ready Queue which contain all the tasks which are being released and are ready to execute. All the tasks in the Ready Queue are sorted according to the decreasing order of their priority. b)a Candidate Queue where all the candidate tasks whose priority is greater than or equal to the priority of the highest priority task are stored. We iterate over all such tasks in the Ready Queue until we get any task whose value of v becomes less than or equal to 0. From the candidate tasks in the Candidate Queue, we randomly select one task and allow it to execute. The task executes till the next scheduling decision is taken. The scheduler works if a new task arrives, or some executing task completes its execution or the value of v of some task becomes 0.

A. Implementation

The first part of the implementation is generation of some random tasksets. I have generated some random tasksets in Python. I applied the schedulability test on those tasksets and selected only those tasksets which have more than 3 tasks in it. The schedulability test is based on the worst-case response time analysis formula (1). I have implemented the TaskShuffler on ZedBoard Zynq-7000 ARM/FPGA SoC Development Board. ZedBoard is a low-cost development board for the Xilinx Zynq-7000 All Programmable SoC. I used FreeRTOS [2] as my operating system. In my implementation, the TaskShuffler scheduler is implemented as a task. The TaskShuffler scheduler is runs at "tskIDLE_PRIORITY + 1", i.e., priority greater than the "tskIDLE_PRIORITY" which is the Idle Task Priority or the lowest priority that can be assigned to any job in FreeRTOS. The TaskShuffler scheduler runs whenever a new job of a task arrives or an executing job of a task completes its execution or when the value of v of a particular job becomes less than or equal to 0. After running the algorithm, the TaskShuffler randomly selects a job from the set of eligible jobs in the Candidate Queue. The selected job now gets instantiated using the "xTaskCreate" function already defined in FreeRTOS. The job will now be created. FreeRTOS only allows one job to execute at a particular time instant. Hence, on creating the new job, I increment the priority of the newly created job to "tskIDLE_PRIORITY + 2" and allow that job to execute. Since, I need to run the scheduler and the newly created job in simultaneously, so at every cycle the job is suspended using "vTaskSuspend" and the scheduler updates the v value of the rest of the jobs pending in the Ready Queue and checks whether it needs to run the TaskShuffler at that instant. The task is again resumed at the beginning of next cycle using "vTaskResume" function available in FreeRTOS. If the scheduler needs to schedule at any instant, then a context

switch occurs and the executing job is suspended and brought into the Ready Queue. If there are k tasks in the taskset, I can have atmost $k + 1$ jobs in my implementation at a particular instant.

V. Experiments and Results

I have implemented three versions of the scheduler. The first version of the scheduler generates the schedule with no randomization. The second version of the scheduler schedules the tasks by randomly selecting the tasks from the Candidate Queue. The third version of the scheduler implements fine-grained switching of the tasks which is given in Algorithm 1. The fine-grained switching generates a random time quantum and runs the scheduler when any one of the following conditions gets satisfied. The conditions are-

1) the time quantum expires
2) any new task arrives
3) any executing task completes its execution
4) Value of v of any task becomes less than or equal to 0

I have generated the results over a set of 4 tasks with a hyper-period of 40 cycles. Table V shows the schedule generated by a scheduler when no randomization is done when selecting a task. So the schedule remains the same over every hyper-period. Table V shows the schedule when TaskShuffler applies randomly selects the tasks from the Candidate Queue. Table V shows the schedule when fine-grained switching of tasks is applied on the tasks. The implementation of all the three versions of the scheduler are available at [https://github.com/anki2911/RTOS_PROJECT-.git].

VI. Conclusion

Though some slots are more likely for most of the tasks, yet on applying randomization on the tasks, the schedule becomes less predictable and hence it becomes difficult for the adversary to predict the schedule for launching timing attacks.

References

[1] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] R. Barry. *FreeRTOS reference manual: API functions and configuration options*. Real Time Engineers Limited, 2009.

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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				

TABLE I

SCHEDULE FOR A HYPER-PERIOD OF 40 CYCLES OVER A TASKSET OF 4 TASK WITHOUT RANDOMIZATION

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
T0	T2	T2	T2	T1	T0	T1	T1	T1	T1	T1	T0	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T2	T2	T2	T1	T0	T1	T1	T1	T1	T0	T2	T1	T3	T3	T0	T1	T1	T1			T0	T2	T2	T2	T1	T1	T1	T0	T2		T0		T1	T1	T1	T0			
T3	T1	T1	T1	T0	T0	T2	T2	T2	T2	T0	T3	T1	T1	T1	T0	T1	T1	T1		T0	T2	T2	T2	T1	T1	T1	T0	T2		T0	T1	T1	T1	T0					
T0	T1	T1	T1	T2	T0	T2	T2	T1	T1	T1	T0	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T0	T2	T2	T2	T1	T0	T1	T1	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T1	T1	T0	T2		T0		T1	T1	T1	T0				

TABLE II

SCHEDULE FOR A HYPER-PERIOD OF 40 CYCLES OVER A TASKSET OF 4 TASK WITH RANDOMIZATION

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
T0	T1	T2	T2	T2	T0	T1	T1	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T1	T1	T0	T2		T0		T1	T1	T1	T0				
T0	T1	T1	T2	T2	T1	T0	T2	T1	T1	T1	T0	T2	T3	T3	T0	T1	T1	T1		T0		T2	T2	T2	T1	T1	T0	T1	T2	T0		T1	T1	T1	T0				
T0	T1	T2	T1	T1	T0	T2	T2	T1	T1	T0	T1	T2	T3	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T1	T1	T0	T2		T0			T1	T1	T0	T1			
T1	T1	T1	T0	T2	T0	T2	T2	T1	T1	T1	T0	T3	T2	T3	T0	T1	T1	T1		T0	T2	T2	T2	T1	T0	T1	T1	T2		T0		T1	T1	T1	T0				
T2	T2	T1	T2	T0	T0	T1	T1	T1	T1	T0	T1	T3	T3	T2	T0	T1	T1	T1		T0	T2	T2	T2	T1	T1	T1	T0	T2		T0		T1	T1	T1	T0				

TABLE III

SCHEDULE FOR A HYPER-PERIOD OF 40 CYCLES OVER A TASKSET OF 4 TASK WITH FINE-GRAINED SWITCHING