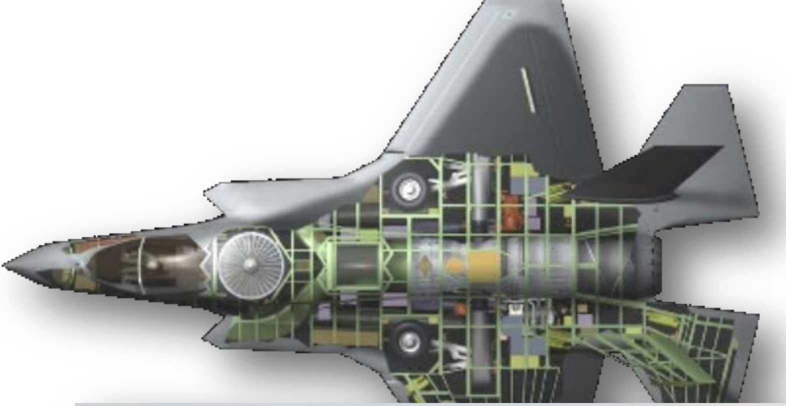
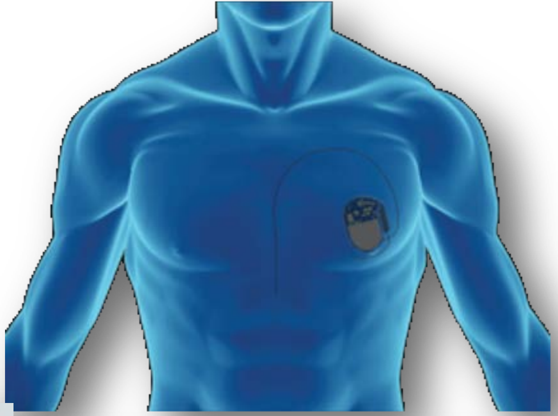
“Who has the time?”

The interplay of Timing and Resiliency in Cyber-Physical Systems

SIBIN MOHAN

DEPT. OF COMPUTER SCIENCE, DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

sibin@illinois.edu


ns [C




PS

Design Challenges

Limited Resources
- Computational power, energy, cost

Timing Requirement
- Safety, reliability, deadlines

System Upgrade
- Verifiability

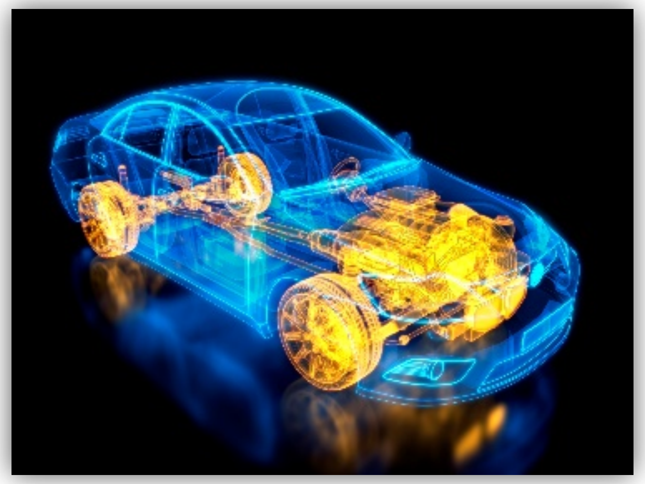
CPS Constraints

- Many CPS have **real-time** constraints

  "requires both, logical correctness as well as **temporal** correctness"
- Temporal correctness defined as a constraint: **deadline**
- Deadlines determine usefulness of results
  - deadline passes → usefulness drops
- Use **well-defined scheduling algorithms**

- *E.g.*: Anti-lock Braking System (ABS) in modern automobiles
  - must function correctly in **milliseconds** time-frame
  - even 1 second might be too late

  (e.g.: a car traveling at 60 mph has travelled 88 ft. in 1s!)



Understanding timing behavior is critical

Physically isolated

Specialized protocols & hardware

Not connected to the internet

Attacks on **Industrial Control Systems** [Stuxnet!]

Hijacking of **automotive** systems

CPS SECURITY [?]

Limited capabilities

Finite (often severely constrained) resources

Vulnerabilities in implantable (and other) **medical** devices

Vulnerable **avionics** systems

Power grids & other utilities

First, we need to understand vulnerabilities in CPS

Today's Talk
[RTSS 2016, ECRTS 2017, DATE 2018, RTAS 2019]

- Challenges to Resiliency of Cyber-Physical Systems (CPS)
  - How to leak critical information from CPS with real-time constraints and
  - Use that information to **break** the CPS

- Integrate mechanisms to detect adversarial actions
  - And still **maintain the integrity** of the CPS

Outline

- **ScheduLeak**: methods to leak schedule information
- **Contego:** Integrate security & maintain real-time requirements

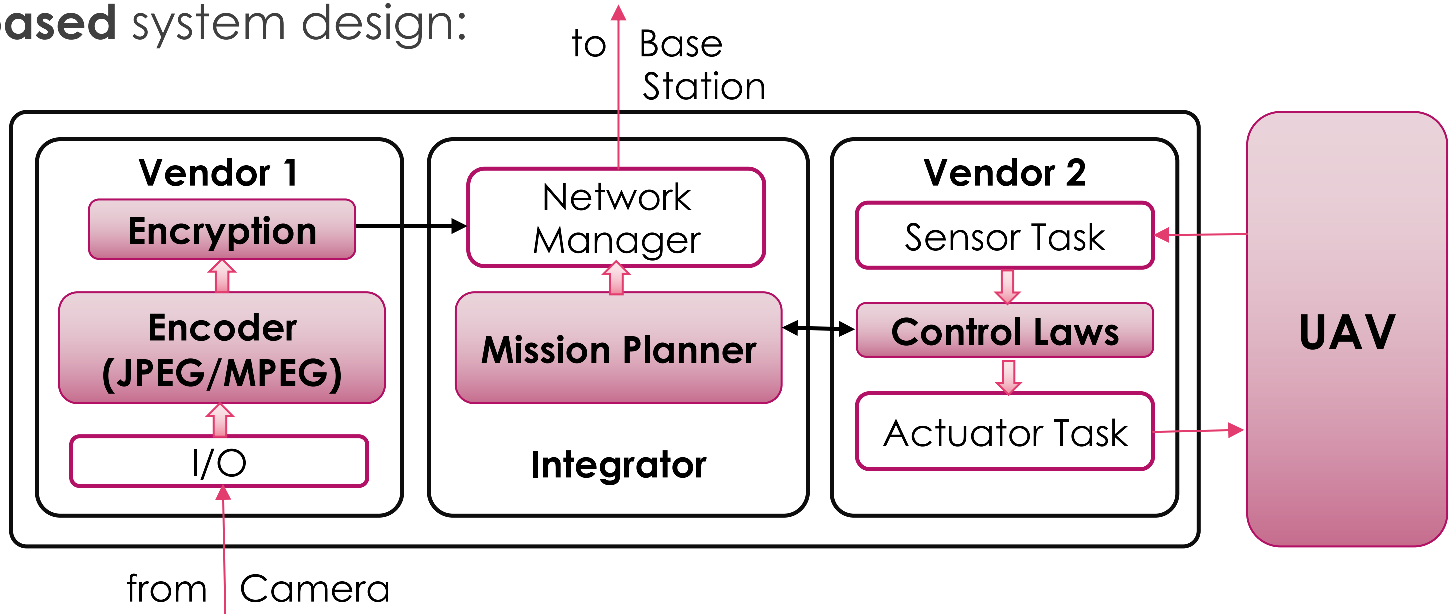
ScheduLeak

- Exfiltration of Critical Information
- Reconnaissance

“given knowledge of the scheduling algorithms used in the system, can we recreate its exact timing schedule?”

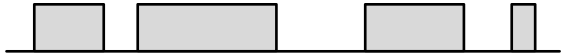
Adversary model & Assumptions

- **Reconnaissance ➔** important step in many security attacks [e.g. Stuxnet]
- Ability to **intrude** into the system **undetected**
- Motivation: **steal information** about system operation/modes/timing information/etc.
  - **User space** activities → as much as possible
- **Vendor-based** system design:



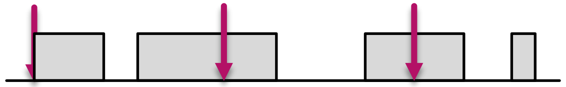
Attack Scenario Overview

There is some schedule (on the victim system)

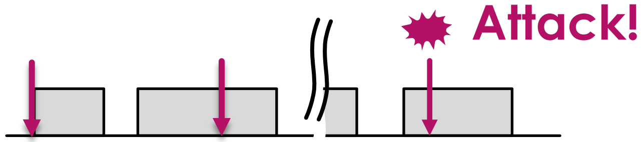


The adversary **observes** and **analyzes** the schedule and **reconstructs** precise timing information

Inferring arrivals of a “victim” task



The attacker can then launch a major attack at a future instant that can **cause the most amount of damage**



Adversary Model [contd.]

- Assumption: Fixed-Priority Real-Time Systems [E.g. RM]
  - Attacker's task (observer task) *periodic* or *sporadic*
  - Victim task *periodic*
  - Other tasks *periodic* or *sporadic*
- Requirements
  - The attacker knows the victim task's period
  - The **observer** task has **lower priority** than the victim task
- Attack Goals
  - **Predict the victim task's future arrival points in time**

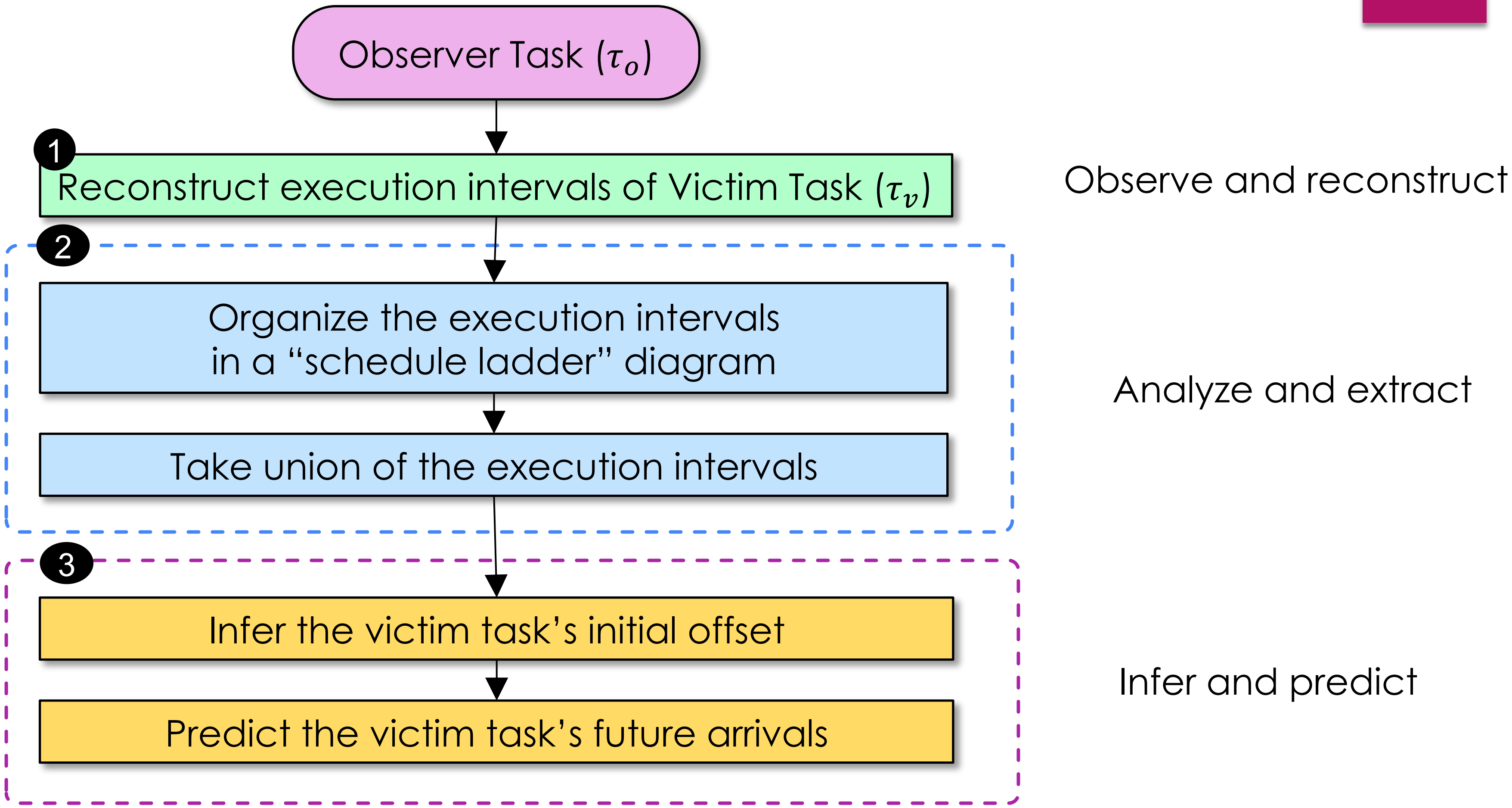
Real-Time Tasks

- **Periodic**
  - Jobs released periodically
  - Relative deadlines
- **Sporadic**
  - Release/arrival times specified
  - Inter-arrival times
  - Absolute deadlines

worst-case execution times

[RTAS 2019] Chen et al., A Novel Side-Channel in Real-Time Schedulers.

ScheduLeak Attack



ScheduLeak Algorithms

Observer task has **lower priority** than victim task

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

Observer Task τ_o Other Tasks

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

1 Reconstruct execution intervals of τ_v

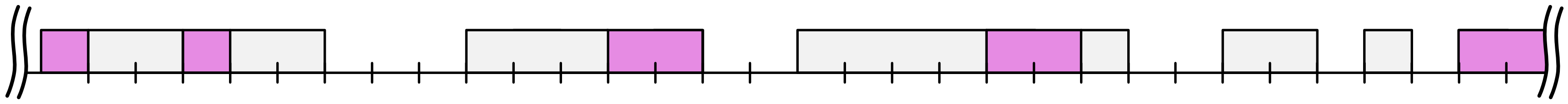
Observer Task τ_o Other Tasks

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

1 Reconstruct execution intervals of τ_v

System Schedule Ground Truth:



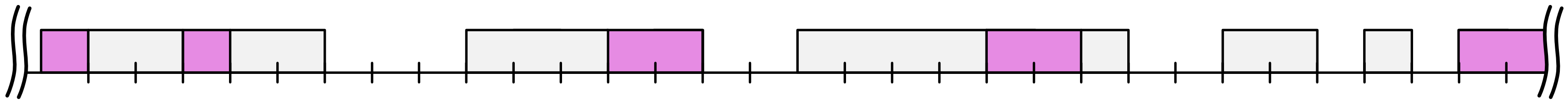
Observer Task τ_o Other Tasks

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

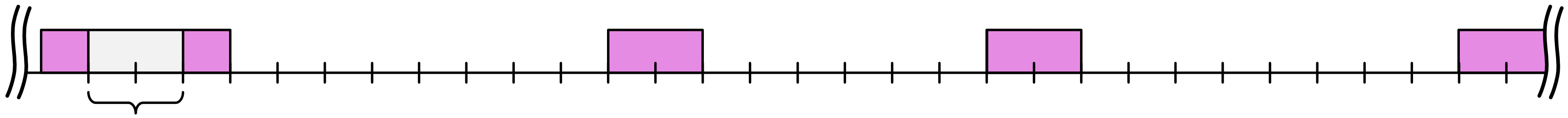
1 Reconstruct execution intervals of τ_v

System Schedule Ground Truth:



What the attacker can observe

Execution Intervals Reconstructed by the Observer Task:



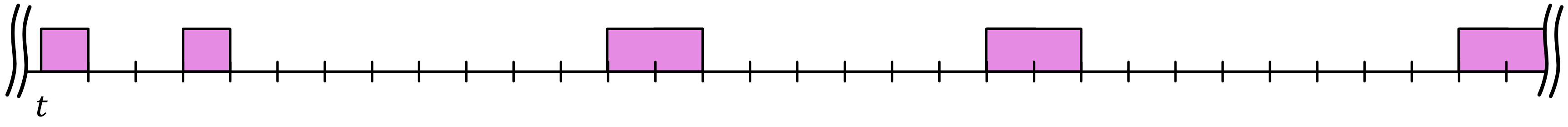
Some tasks preempted the observer task

Observer Task τ_o | Other Tasks

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

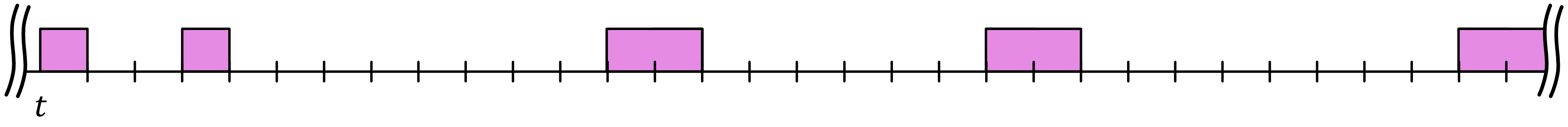
2 Organize the execution intervals in a "**schedule ladder diagram**"



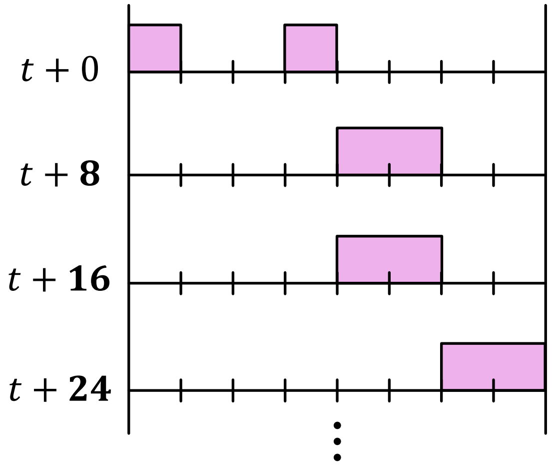
ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	**8**	2
Task 4	6	1

2 Organize the execution intervals in a "**schedule ladder diagram**"



Place the intervals in a ladder diagram
(width equals the victim task's period)



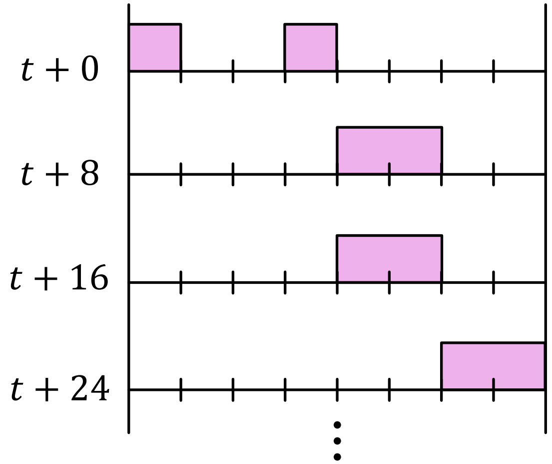
Still dealing with the Observer task executions

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	**8**	2
Task 4	6	1

2

Organize the execution intervals in a "**schedule ladder diagram**"



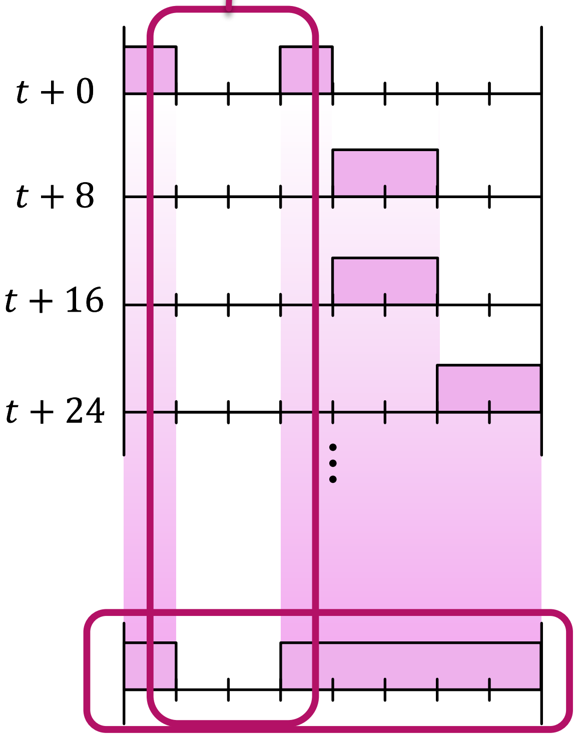
ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
	6	1

2. Organize the execution intervals in a "**schedule ladder diagram**"

Take union of the execution intervals

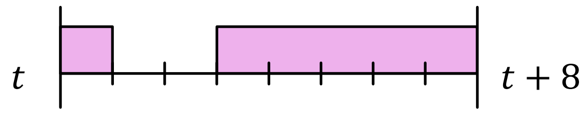
No observer task OR any other lower priority task here



ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

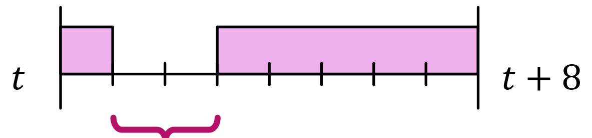
3 Infer the victim task's initial offset



ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

3 Infer the victim task's initial offset



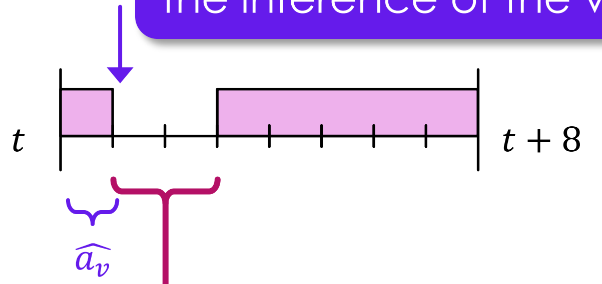
Tasks with lower priorities (e.g. observer task) **cannot** appear in this column!

ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

3 Infer the victim task's initial offset

We take the **starting point** of the empty column as the inference of the victim task's initial offset.



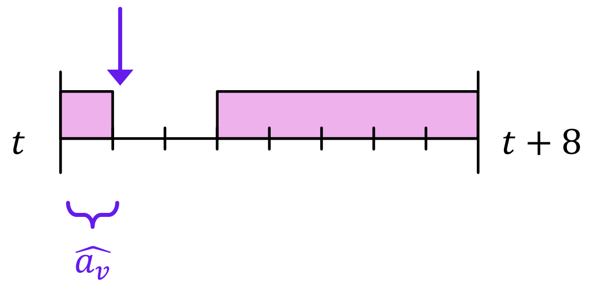
Tasks with lower priorities (e.g. observer task) **cannot** appear in this column!

ScheduLeak Algorithms

3 Infer the victim task's initial offset

Predict the victim task's future arrivals

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

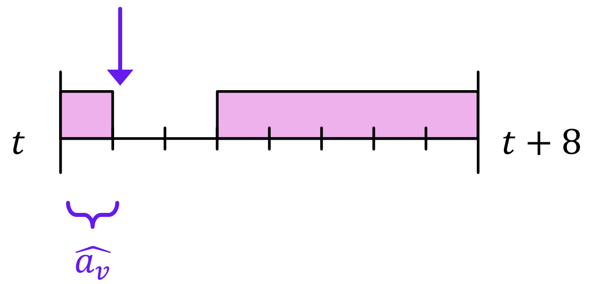


ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

3 Infer the victim task's initial offset

Predict the victim task's future arrivals



The **victim task's future arrival times** can be computed by

$$t + \widehat{a_v} + p_v + T$$

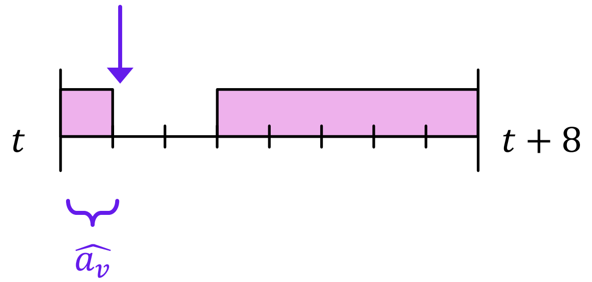
ScheduLeak Algorithms

Task ID	Period	Exec Time
Observer Task	15	1
Task 2	10	2
Victim Task (τ_v)	8	2
Task 4	6	1

3 Infer the victim task's initial offset

↓

Predict the victim task's future arrivals



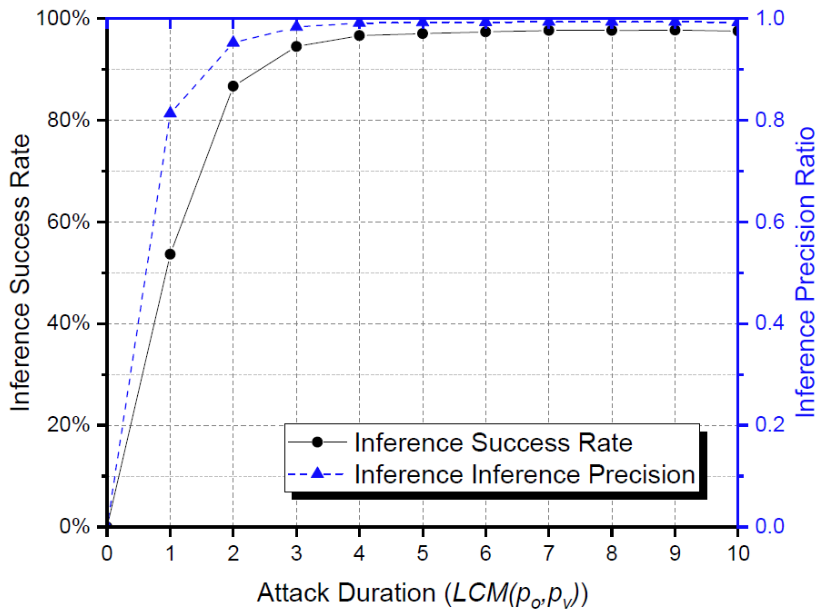
The **victim task's future arrival times** can be computed by

$$\underbrace{t}_{\text{ladder diagram starting point}} + \underbrace{\widehat{a_v}}_{\text{Inferred victim task's Initial offset}} + \underbrace{p_v}_{\text{victim task's period}} + \underbrace{T}_{\text{desired arrival number}}$$

Can predict, with high precision, arrival times of victim!

Experimental Results

Duration of Observations



Success rate and precision ratio are stabilized after $5 \cdot LCM(p_o, p_v)$

- Success rate: **97%**
- Precision ratio: **0.99**

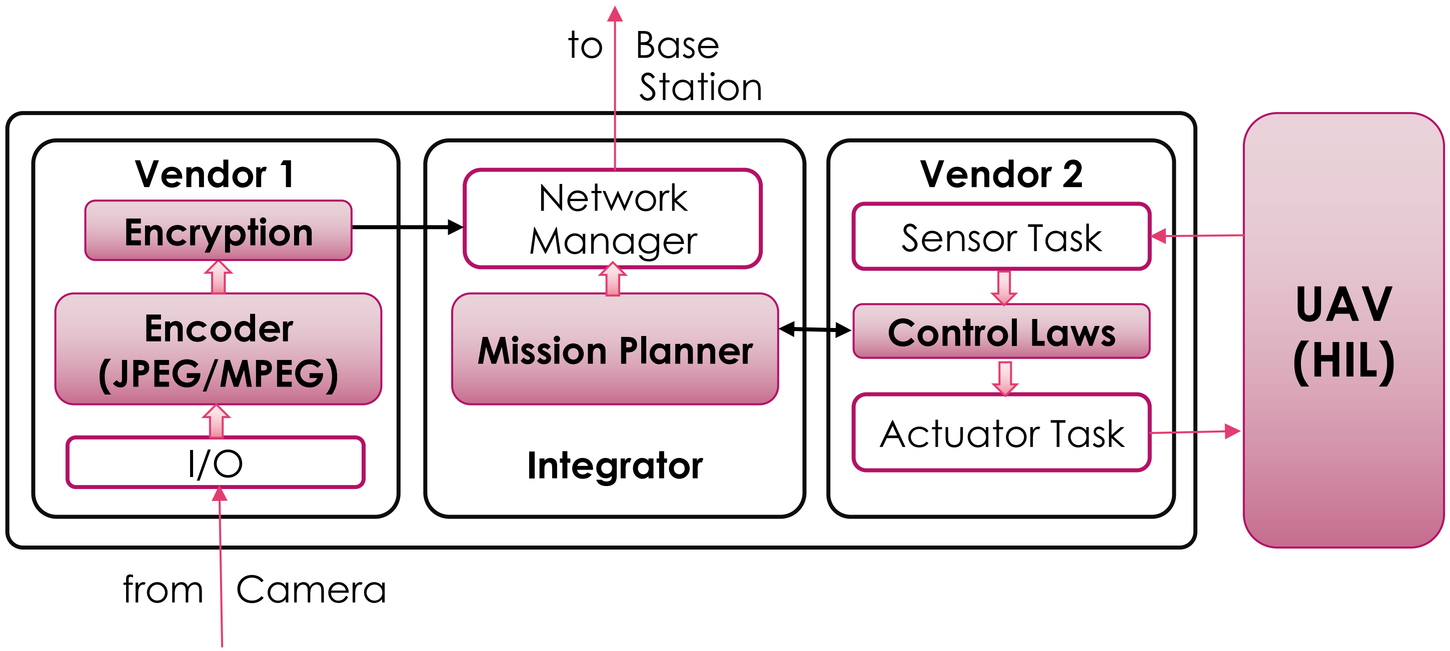
Note

1. Each data point represents the mean of **12000 tasksets** for the given observation duration
2. **Inference Success Rate**: an inference is successful if attacker is able to exactly infer the victim task's initial offset
3. **Inference Precision Ratio**: the ratio of how close the inference to the true initial offset

What can we do with information gleaned using ScheduLeak?

Demonstration 1
Cache-Timing Side-Channel Attack

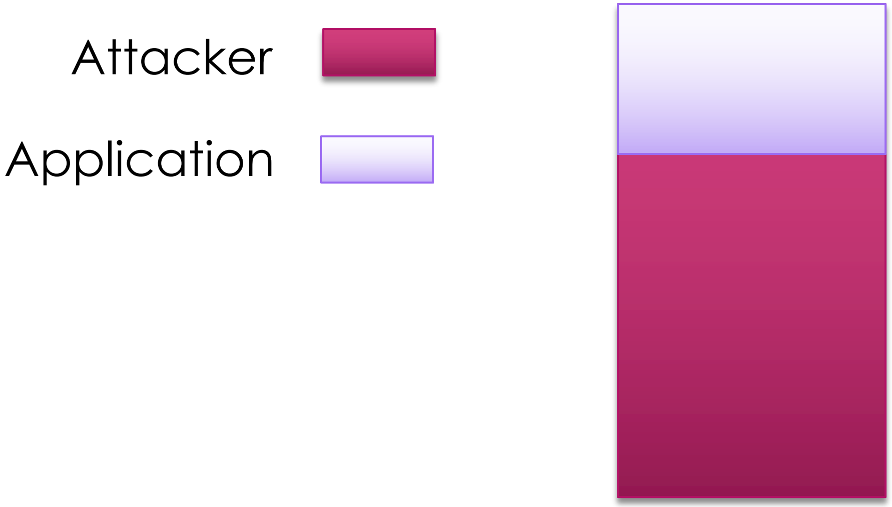
- UAV that flies across several locations
  - **High resolution pictures of points of interest**
  - Low resolution otherwise
- Image processing task
  - **Victim** task



Cache timing Attack model

- Timing attacks

"attacker attempts to **steal the information** from the system by analyzing time variation of a function"



1. Attacker fills the cache

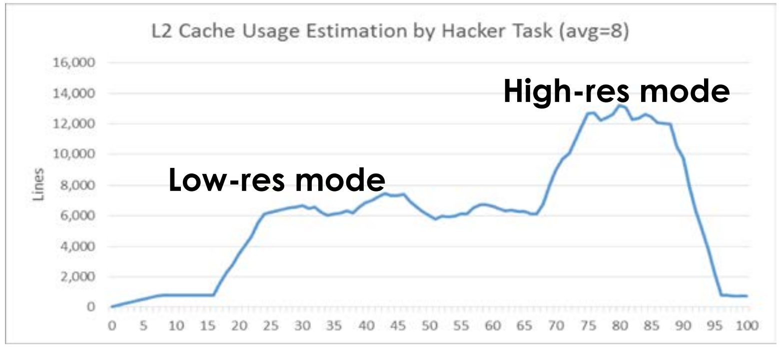
2. Let application use cache

3. Attacker measures **cache-miss** and **cache-hit** ratio to gauge the cache usage

- Well known in security and system literature
  - Steal cryptographic keys, snooping in cloud computing, etc.

Demonstration 1
Cache-Timing Side-Channel Attack

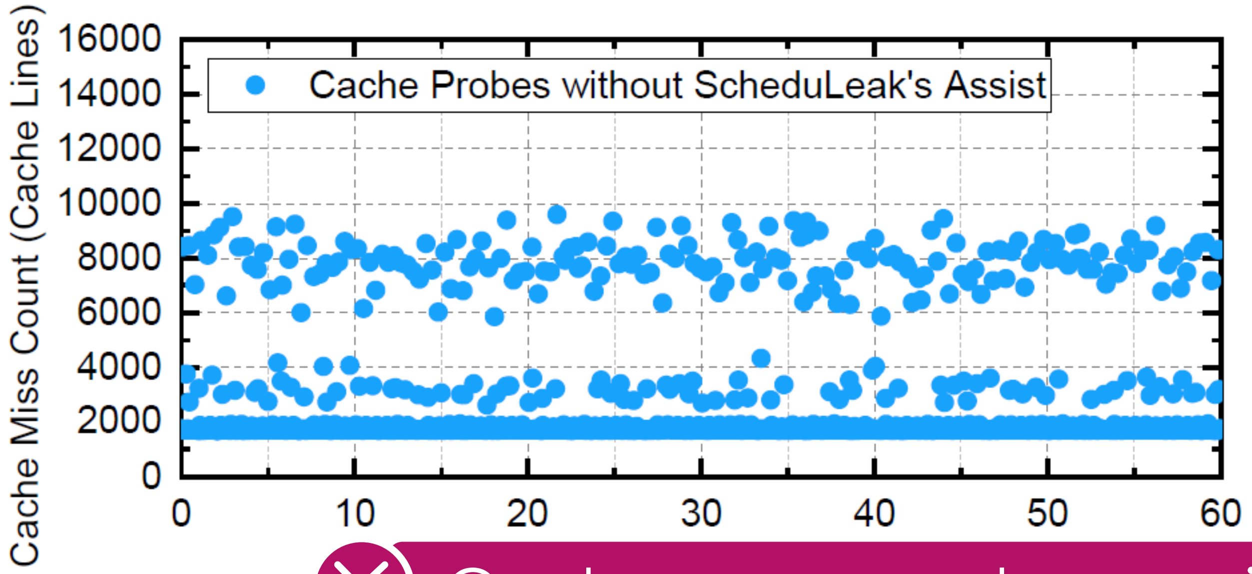
- Attack Goals:
  - Probe (coarse-grained) memory usage of victim task
  - Recover locations of interest → points where memory usage (of victim task) is high



Measurements on Xilinx Zedboard Zynq-7000, FreeRTOS, [CPU Freq: 666MHz, L2 Cache: 512KB, 32 byte line size]

Demonstration 1
Cache-Timing Side-Channel Attack

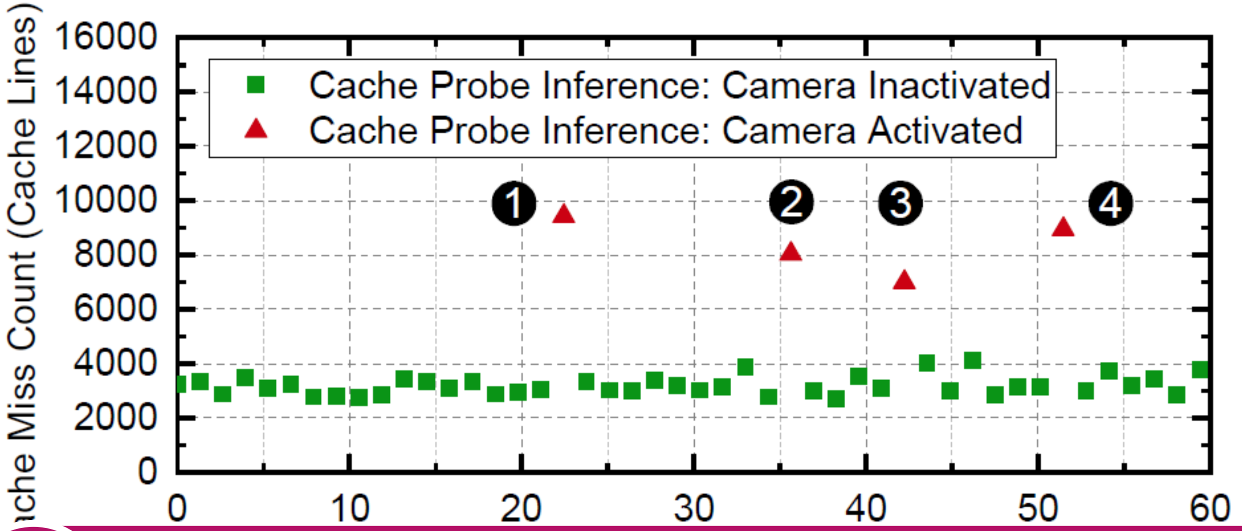
- **Without** ScheduLeak–based information
  - Attackers are forced to **randomly sample** the system
  - To detect memory usage changes



Cache usage probes are indistinguishable

Demonstration 1
Cache-Timing Side-Channel Attack

- **With precise timing information** from ScheduLeak
  - Attackers can launch cache-timing attack at more precise points
  - **Very close to the execution of the victim task**



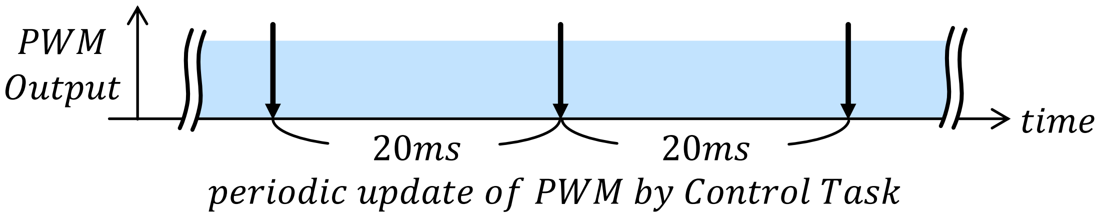
Four locations are recovered from the cache usage probes

Demonstration 2

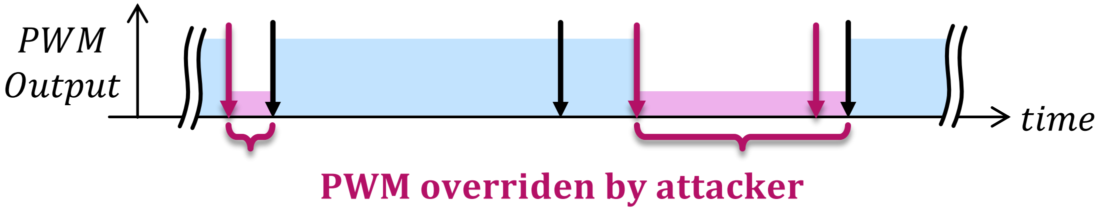
Interference with Control (Actuation Signals) of CPS

- Autonomous rover/drone that has ESC/servos
  - Control throttle and steering
- PWM control task (victim) updates PWM values periodically
- Attack goals:
  - Interleave PWM signals to **override** control of throttle/steering
  - Cause system to **crash** or worse, **take over control**!

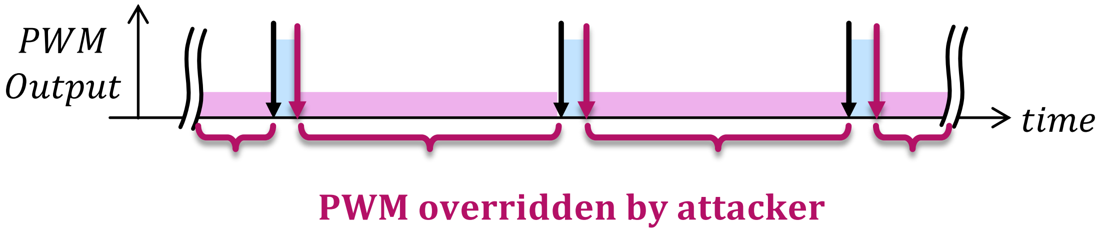
Demonstration 2
Interference with Control (Actuation Signals) of CPS



Without precise knowledge from ScheduLeak



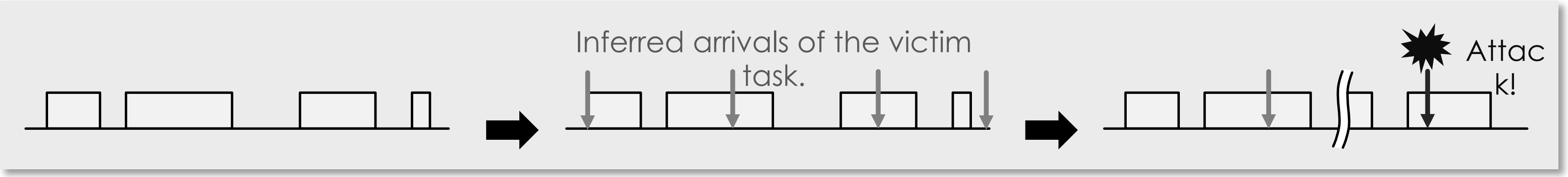
Using information from ScheduLeak



ScheduLeak Demo

ScheduLeak Summary

- Reconnaissance attack algorithms
- Targeting sporadic and mixed real-time CPS
- Stealthy and Effective
- **No root privileges** required for ScheduLeak



More videos [including cache attack demo]: **https://scheduleak.github.io**

Contego

- Integration of Security in Real-Time CPS
- For legacy as well as future systems

"if we are to integrate any (arbitrary) security mechanism/application, can it be done without perturbing the timing guarantees of the CPS?"

Integrating Security into Legacy CPS

- Integration into **Legacy** Real-Time Systems (RTS): ➡ **NOT feasible**
  - Requires major modification of system/task parameters
    - run-times, period, task execution order, etc.

- Security mechanisms need to:
  - **co-exist** with the real-time tasks
  - operate **without** impacting timing & safety constraints of control logic

Integrating Security Tasks
Requirements

- How to integrate security tasks
  - **without perturbing** real-time tasks most of the time?
- How to determine the **frequency** of the security tasks?
  - improve **responsiveness** of security mechanisms?

Examples of Security Tasks [from Linux]

Security Tasks	Function
Check own binary [Tripwire]	Scan files in the following locations: /usr/sbin/siggen, /usr/sbin/tripwire, /usr/sbin/twadmin, /usr/sbin/twprint
Check critical executables [Tripwire]	Scan file-system binary (/bin, /sbin)
Monitor network traffic [Bro]	Scan predefined network interface(en0)

Performance Criteria

1. **Frequency of Monitoring**: if monitoring interval is
   - too large ➔ **delays detection** of adversary
   - too short ➔ **impacts schedulability** of real-time tasks

Performance Criteria [contd.]

2. **Responsiveness**: when a security breach is suspected:
   - security routine may be required to switch to **more active role**
     - more fine-grained checking
     - restart/reload from trusted copy
     - graceful degradation
     - cleanup tasks
     - raise alarms
     - etc.

Proposed Approach: Overview

- Add **additional fixed-priority sporadic security tasks**
  - Any one of protection, detection or response mechanisms
  - Example: Tripwire, Bro, OSSEC, etc.

Initial Approach
[RTSS 2016]

- Ensure security **without perturbing** real-time scheduling order
  - Execute security tasks as **lowest priority tasks**
  - **Slower response times ➔** from security/monitoring perspective

[RTSS 2016] Hasan *et al*., Exploring opportunistic execution for integrating security into legacy hard real-time systems.

Can we do better?

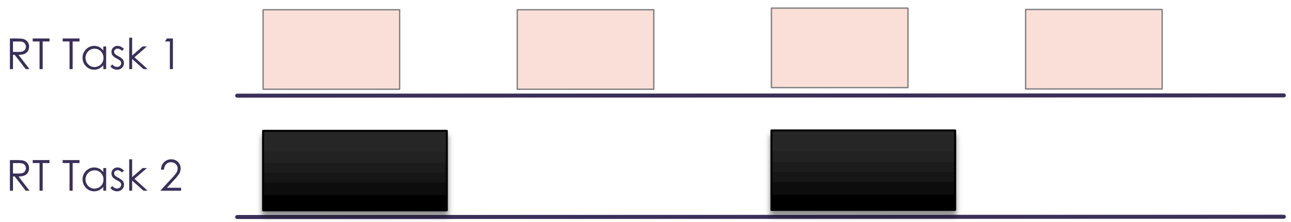
Contego
[ECRTS 2017]

- Allow security tasks to run in two modes:
  - **PASSIVE**
  - **ACTIVE**

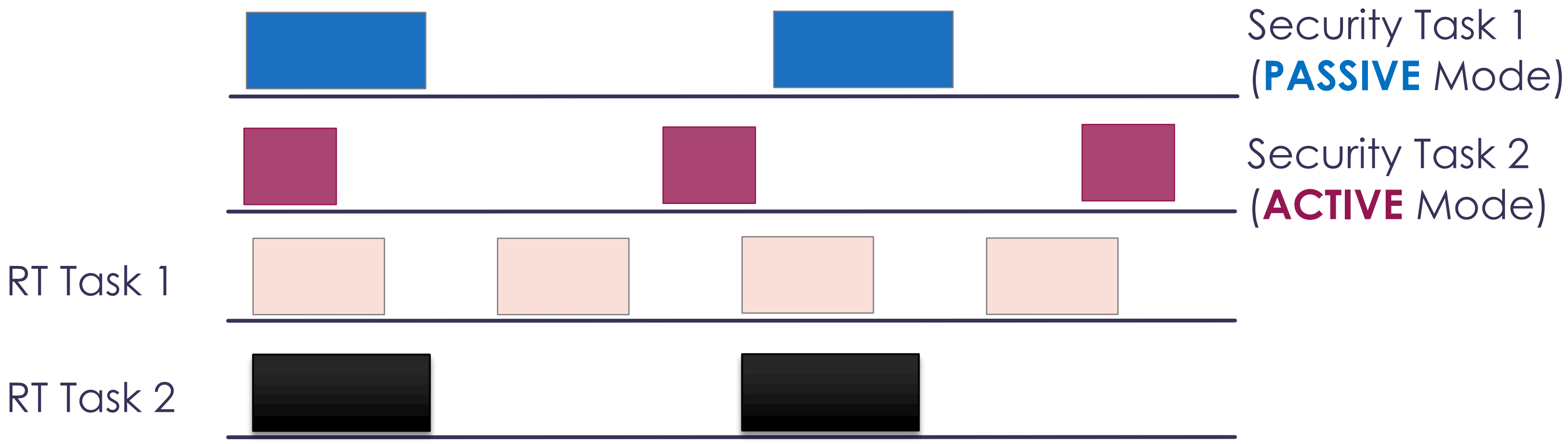
Contego

- Allow security tasks to run in two modes:
  - **PASSIVE**
    - Execute opportunistically with lowest priority
  - **ACTIVE**
    - Switch to other (active) mechanisms if abnormality is detected

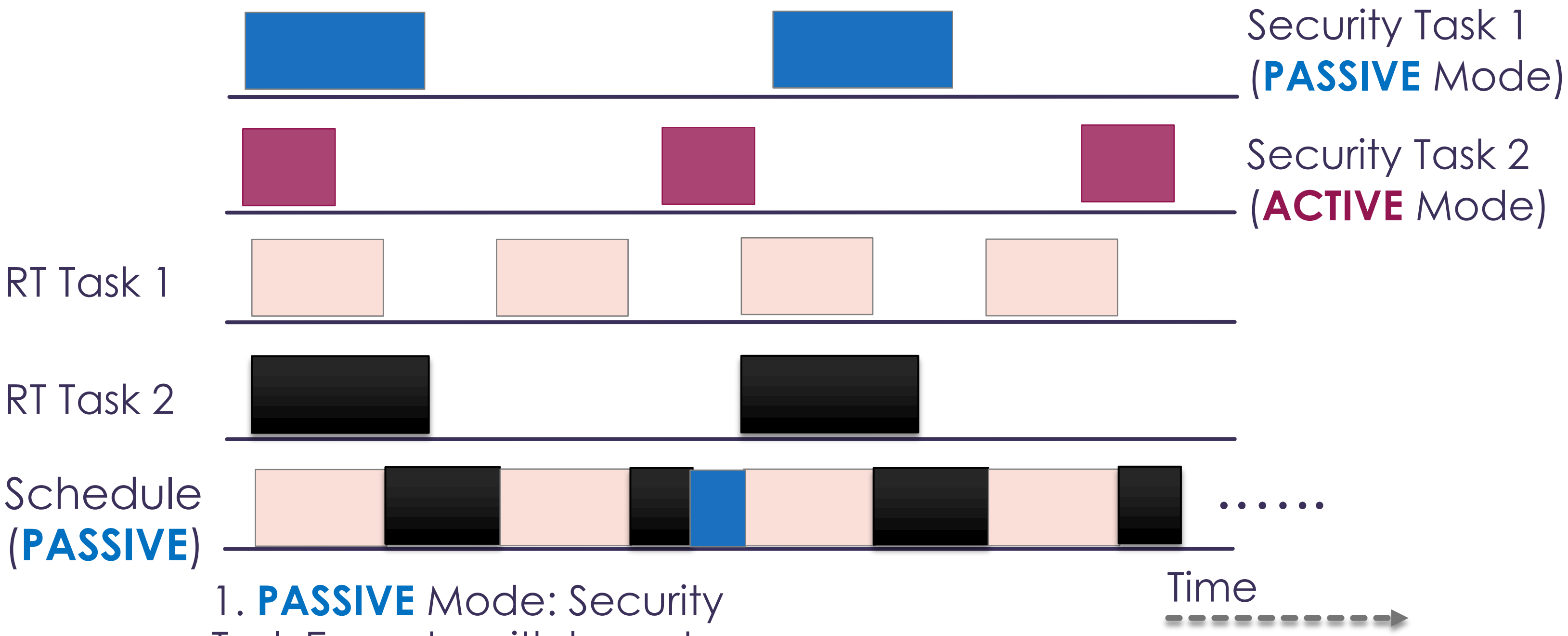
Contego Example



Contego Example

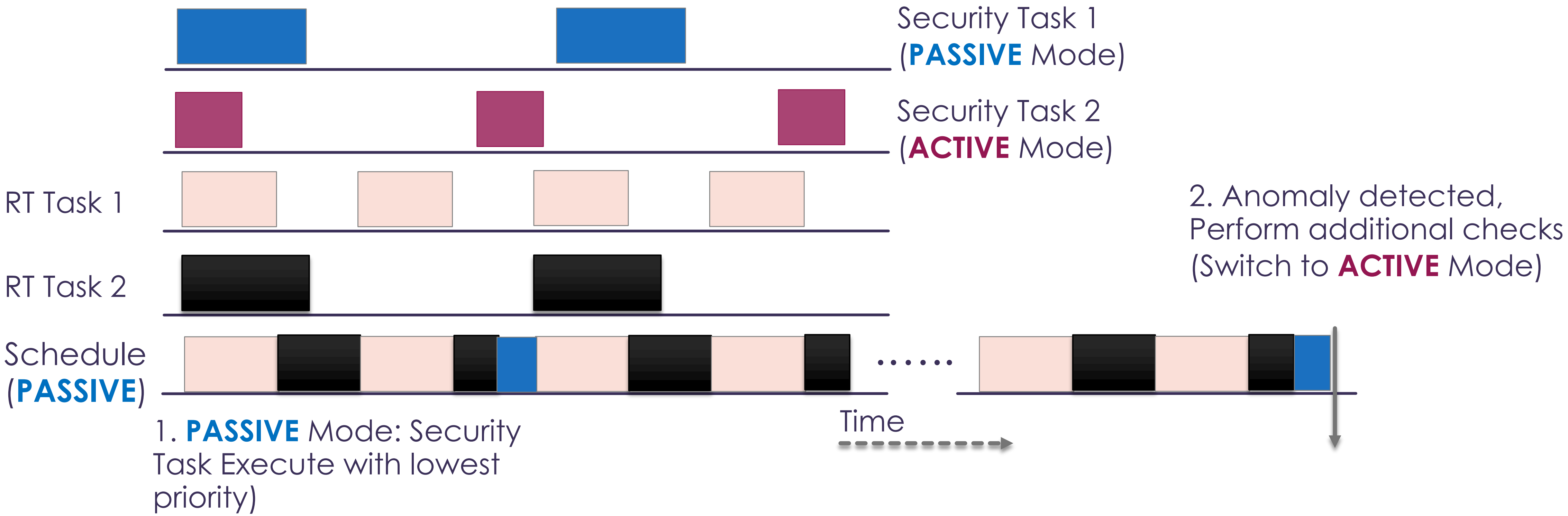


Contego Example

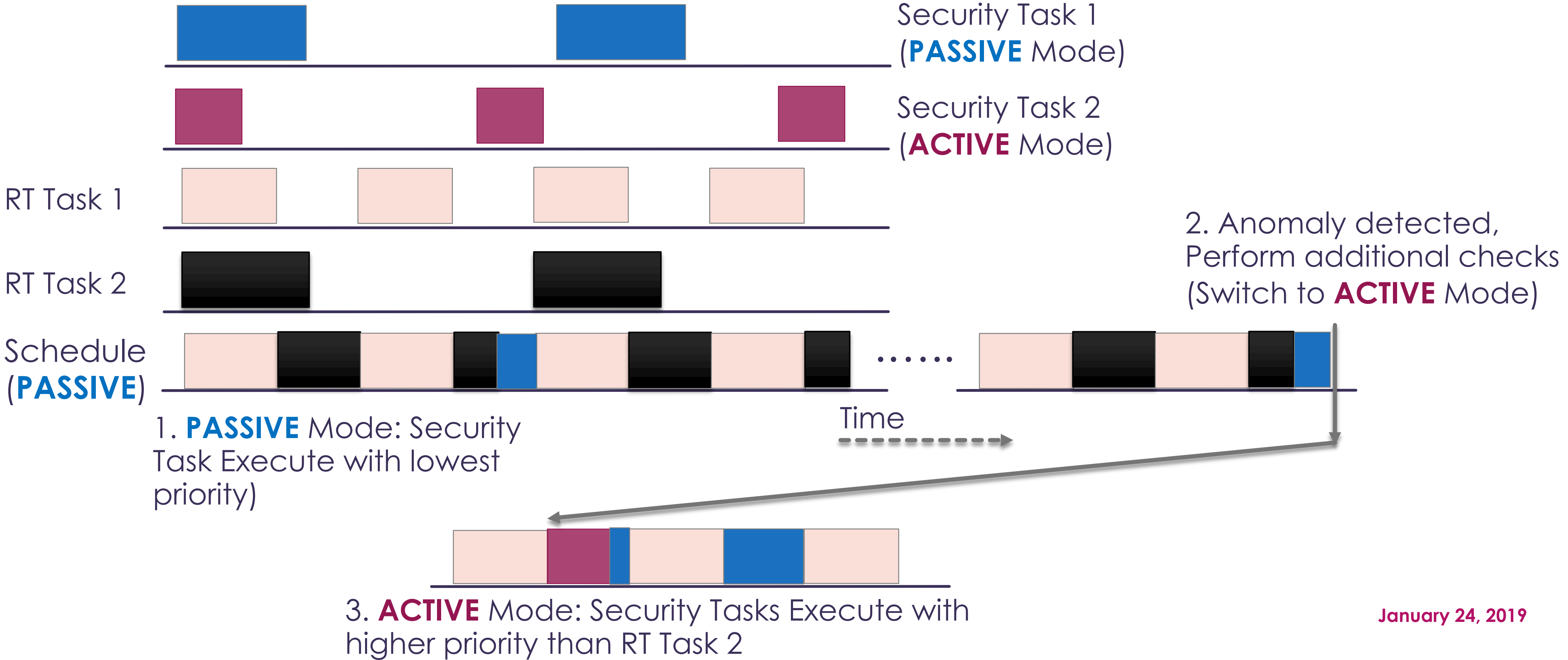


1. **PASSIVE** Mode: Security Task Execute with lowest priority)

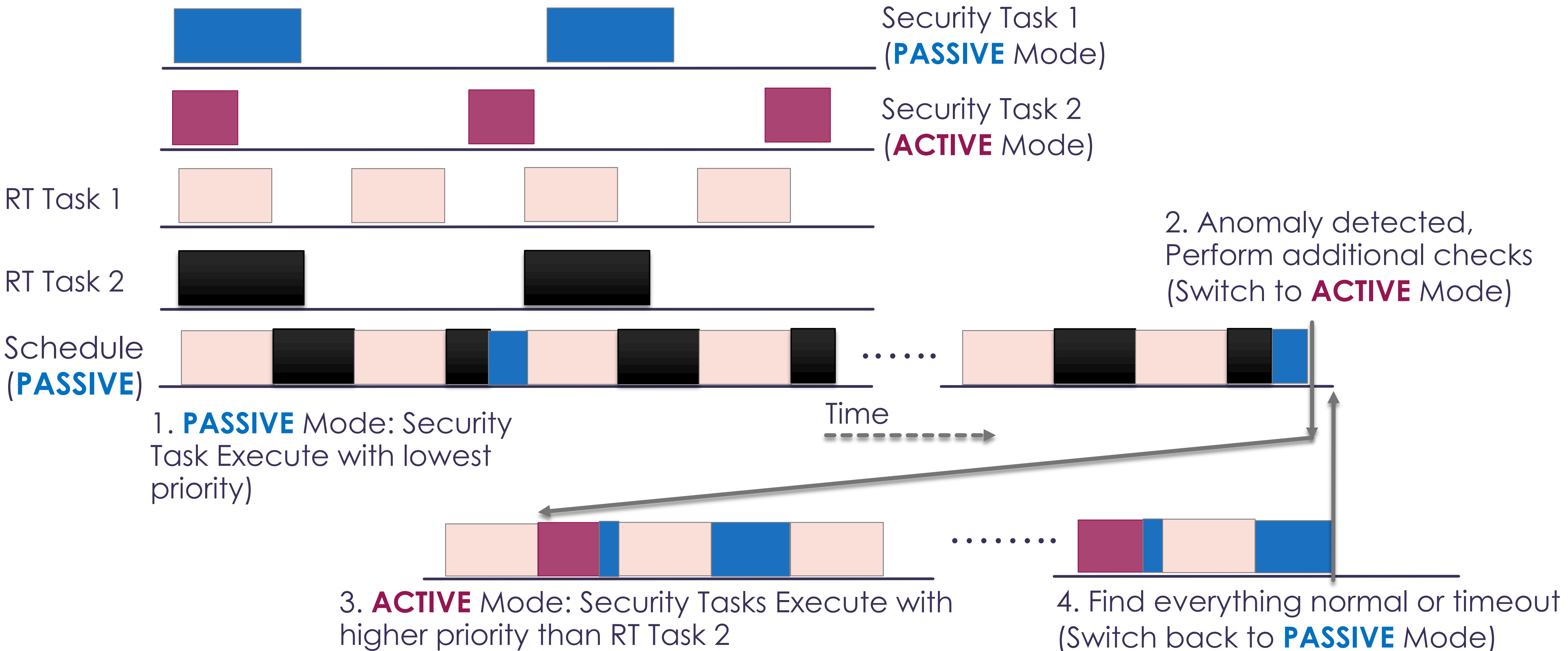
Contego Example



Contego Example



Contego Example



System Model

- Fixed-priority uniprocessor system
  - Implicit deadlines
  - Follows Rate Monotonic order
  - ‘m’ Real-time tasks → ‘m’ distinct priority-levels
- Security tasks are characterized by $(C_i, T_i^{des}, T_i^{max}, \omega_i)$
- **No specific assumptions about the security tasks** in both modes
  - May contain completely different tasks
  - or (partially) identical tasks with different parameters

System Model [contd.]

- **PASSIVE** mode:
  - Security tasks are executed with **lower priority** than the real-time tasks
- **ACTIVE** mode:
  - Security tasks can execute in **any** priority-level between $[l_S, m]$
    - Recall 'm': number of real-time priorities
    - 'l_s': upper limit for priorities of active security tasks

Problem Description

- Metric: Tightness of achievable periodic monitoring

$$\eta_i = \frac{T_i^{des}}{T_i}$$

- Any period within $T_i^{des} \leq T_i \leq T_i^{max}$ is acceptable
- **Actual period** T_i is unknown (for **PASSIVE** and **ACTIVE** modes)
- **Priority levels** are unknown (For **ACTIVE** mode)

Solution

Constrained Optimization Problem [ECRTS 2017]

- Formulate as a **constrained optimization problem**

For **PASSIVE** mode:

`Maximize` **`Tightness`** subject to:

a. `The system is schedulable`

b. `Security tasks periods > real-time task periods`

c. `Security tasks' periods are within acceptable bound`

$$\max_{\mathbf{T}^{pa}} \quad \eta^{pa}$$

$$\text{Subject to:} \quad \sum_{\tau_i \in \Gamma_S^{pa}} \frac{C_i}{T_i} \leq (m + n_p)(2^{\frac{1}{m+n_p}} - 1) - \sum_{\tau_j \in \Gamma_R} \frac{C_j}{T_j}$$

$$T_i \geq \max_{\tau_j \in \Gamma_R} T_j \qquad \forall \tau_i \in \Gamma_S^{pa}$$

$$T_i^{des} \leq T_i \leq T_i^{max} \qquad \forall \tau_i \in \Gamma_S^{pa}$$

[ECRTS 2017] Hassan *et al.* Contego: An Adaptive Framework for Integrating Security Tasks in Real-Time Systems.

Solution

Constrained Optimization Problem [ECRTS 2017]

- Formulate as a **constrained optimization problem**

For **ACTIVE** mode (given a priority-level, l_s):

Maximize **Tightness** subject to:

a. The system is schedulable

b. Satisfy execution order of **higher-priority** RT tasks

$$T_i \geq \max_{\tau_j \in \Gamma_{R_{hp(l_S)}}} T_j, \quad \forall \tau_i \in \Gamma_S^{ac}$$

c. Security tasks' periods are within acceptable bound

[ECRTS 2017] Hassan *et al.* Contego: An Adaptive Framework for Integrating Security Tasks in Real-Time Systems.

Limitations and Solution

- Non-linear constraint optimization problem
- Formulation limited by Rate Monotonic bound (69% Utilization)
- Requires analysis on a per-task basis

- Transformed into non-convex Geometric Programming (GP)
- Reformulate the non-convex GP to equivalent convex form
- Solve using known algorithms (Interior Point method)

[ECRTS 2017] Hassan *et al.* Contego: An Adaptive Framework for Integrating Security Tasks in Real-Time Systems.

Evaluation on Embedded Platform

- Experiment with Security applications
  - Platform: 1 GHz ARM Cortex A8, 512 MB RAM
  - OS: Linux with Xenomai real-time patch

Real-Time Tasks [UAV]	Function	Period (ms)
Guidance	Select reference trajectory (altitude & heading)	1000
Controller	Execute closed-loop control functions	5000
Reconnaissance	Read radar/camera data, collect sensitive information, send data to base control station	10000

Evaluation on Embedded Platform

- Experiment with Security applications
  - Platform: 1 GHz ARM Cortex A8, 512 MB RAM
  - OS: Linux with Xenomai real-time patch
  - Security applications: Tripwire, Bro

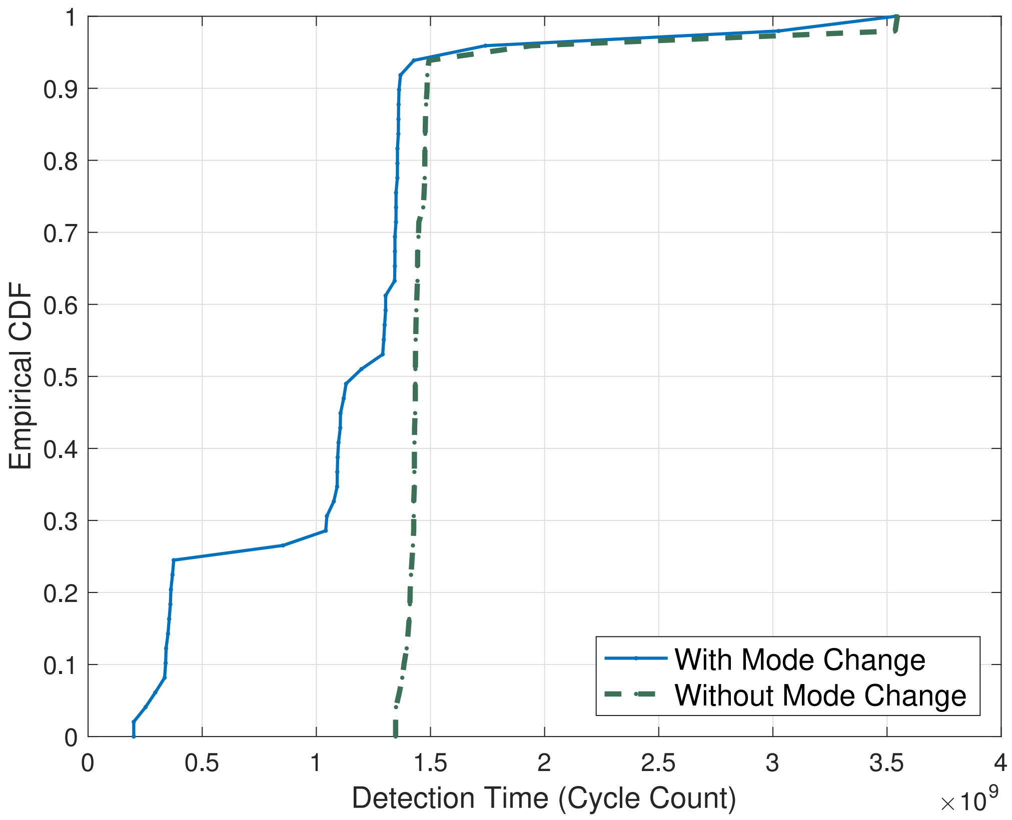
Security Tasks	Function	Mode
Check own binary (Tripwire)	Scan files in the following locations: /usr/sbin/siggen, /usr/sbin/tripwire, /usr/sbin/twadmin, /usr/sbin/twprint	**ACTIVE**
Check critical executables (Tripwire)	Scan binaries in the file-system (/bin, /sbin)	**ACTIVE** and **PASSIVE**
Check Critical libraries (Tripwire)	Scan libraries in the file system (/lib)	**ACTIVE**
Monitor network traffic (Bro)	Scan predefined network interface (en0)	**ACTIVE** and **PASSIVE**

Evaluation on Embedded Platform

- Experiment with Security applications
  - Platform: 1 GHz ARM Cortex A8, 512 MB RAM
  - OS: Linux with Xenomai real-time patch
  - Security applications: Tripwire, Bro

- **Attack** demonstration:
  - Compromise a real-time task
  - Perform network-level DoS attack
  - Also inject shellcodes that modify file-system binary (`/bin`)

Impact on Detection Time



X-axis: CDF of detection time

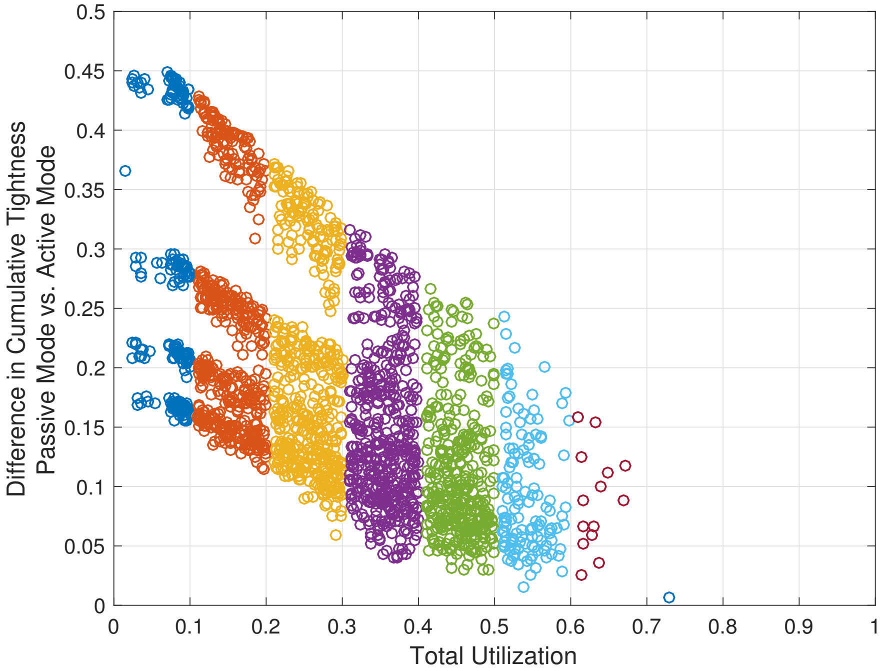
Y-axis: Detection time (cycle count)

Without mode change: Run security tasks with *lowest priority* [RTSS '16]

Contego detects attacks **27.29%** faster than previous scheme

[RTSS '16] Hasan *et al.*, Exploring opportunistic execution for integrating security into legacy hard real-time systems, *RTSS*, 2016

Tightness of Monitoring



X-axis: System Utilization

Y-axis: Difference between tightness $\eta^{av} - \eta^{pa}$

[active mode and passive mode]

Active mode tasks achieve much better tightness than passive mode tasks

- 5000 synthetic task-sets
- Total utilization of Security Tasks: < 30% of the real-time tasks
- I_s upper bounded by 0.4m

Contego Summary

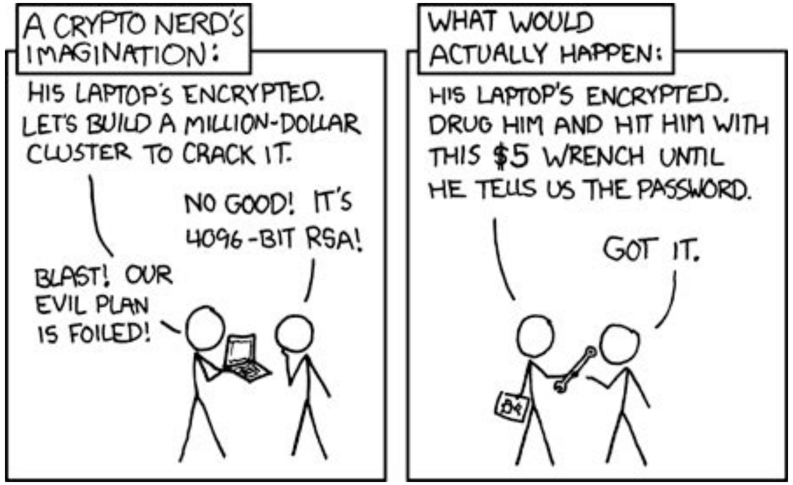
- An **adaptive approach** to integrate security tasks into RTS
- Careful period selection and behavior-based mode switching
  - Improve responsiveness of security mechanisms
  - Retain (most) real-time guarantees
- **Framework for integrating security methods**

Security integration that **maintains** resiliency of real-time CPS

To Summarize…

- Research that explores the **resiliency of cyber-physical systems**
- From both perspectives:
  - How to weaken/break resiliency [ScheduLeak]
  - How to strengthen it [Contego and other work]

Designers of CPS have a better understanding of requirements



Thanks!

- http://sibin-research.blogspot.com
- https://scheduleak.github.io