On the Pitfalls and Vulnerabilities of Schedule Randomization against Schedule-Based Attacks

Mitra Nasri Tam Chantem Gedare Bloom Ryan Gerdes

TU Delft

VIRGINIA TECH

1867 HOWARD UNIVERSITY

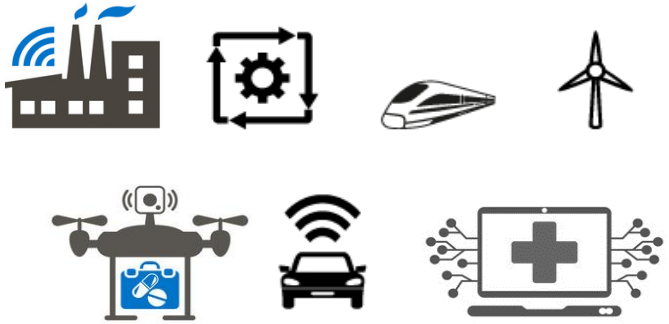
Security
is becoming an important concern for embedded real-time systems

"**Time predictability** makes real-time systems **vulnerable** against **schedule-based attacks**"

Randomize the schedule (while respecting the timing constraints) so that the attacker cannot easily guess when things happen

Electronic devices & wearables

Home automation



Industrial, transport, environmental & healthcare

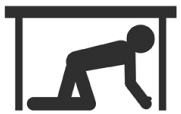
This talk

We **study** the **schedule randomization** as a **security defense** for **real-time systems**

- Is it a **good defense** against **schedule-based attacks**?
- Is it a **good solution** for **real-time systems**?
- How **well** does it **perform** in various **attack scenarios**?

Randomization as a security defense
(in a broader security community)

Information hiding
(probabilistic pseudo-isolation)



Deterministic isolation

Examples
- Address-space layout randomization (ASLR)
- Kernel memory randomization (KLSR)
- Control-flow randomization

In all those use-cases, it was shown that **randomization-based isolations** can be **broken easily and efficiently**

[USENIX'16] "Undermining information hiding (and what to do about it)"

[USENIX'16] "Poking holes in information hiding"

[SP'13] "Practical timing side channel attacks against kernel space ASLR"

[SP'15] "Missing the point(er): On the effectiveness of code pointer integrity"

[CCS'16] "Breaking kernel address space layout randomization with intel TSX"

….

How does schedule randomization work?

Krüger et al. [Krüger et al., ECRTS'18] **Online and Offline**

Offline: build and store randomized schedules for different hyperperiods

Online: use Slot Shifting algorithm to accommodate a randomly chosen task in the schedule

TaskShuffler [Yoon et al., RTAS'16] **Online**

applies a slack stealing method on top of the fixed-priority scheduling to steal the slack of high-priority tasks in favor of a randomly chosen task without jeopardizing schedulability.

K. Krüger, M. Völp, and G. Fohler, "Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems ," in ECRTS, 2018.

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 RTAS, 2016.

Is **schedule randomization** a **good solution to hide** the **schedule-related information?**

We argue that:
it **has far less choices (and entropy)** than other randomization-based defenses such as ASLR

Currently, **there is no VALID security metric** for evaluating schedule randomization
(the existing one is optimistic and not safe)

Guessing the schedule is just the **first part** of an **attack.**

Hence, the right starting point is to **define** schedule-based attacks and their taxonomy.

Contributions

- **Introducing a taxonomy of schedule-based attacks**
- **Study schedule randomization-based defense**
  - Is it a **good defense** against **schedule-based attacks**?
  - Is it a **good solution** for **real-time systems**?
  - How **well** does it **perform** in various **attack scenarios**?
- **Providing a preliminary security test for fixed-priority scheduling**

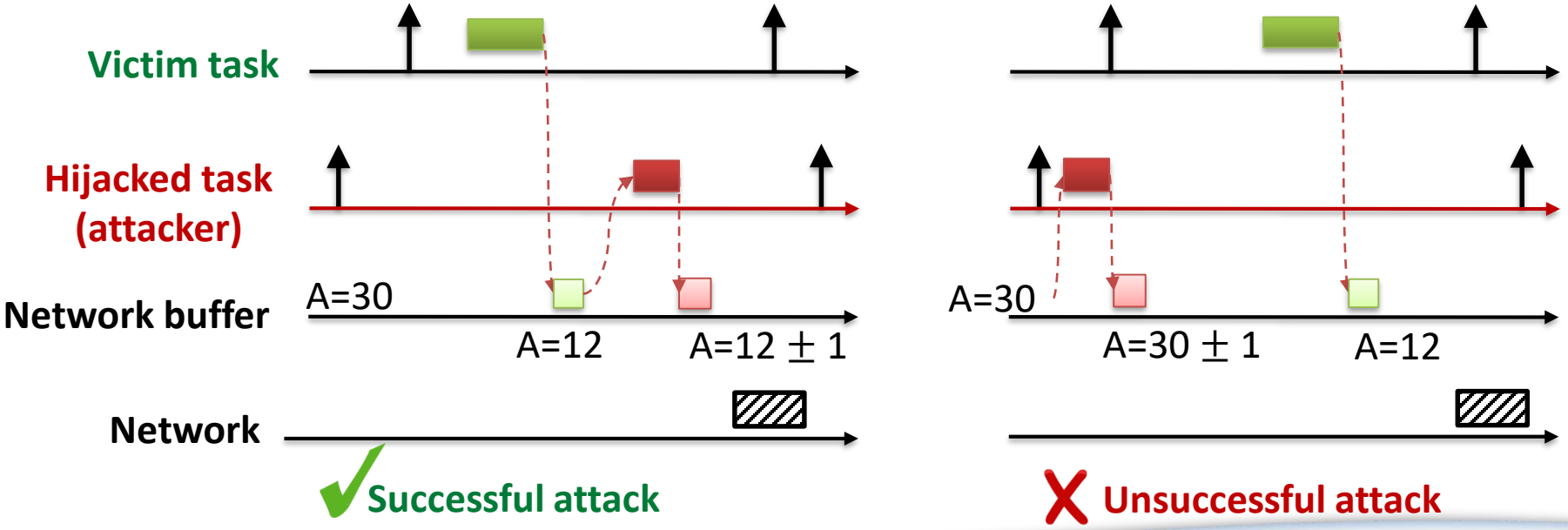
Schedule-based attacks

Attacks whose **success** depend on a *particular ordering* between the **execution window of the attacker** and its targeted task (victim)

Data injection example:

The attacker's goal is to modify the data, e.g., so that the system uses more energy to stabilize or has lower quality of service

The attacker wants to stay stealthy: data modifications are **not distinguishable** from the noise.



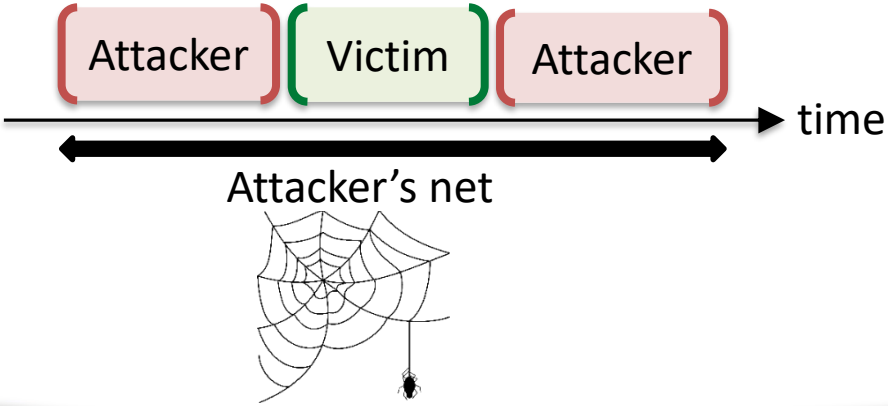
A taxonomy of schedule-based attacks

Anterior attacks | **Posterior attacks** | **Pincer attacks** | **Concurrent attacks**

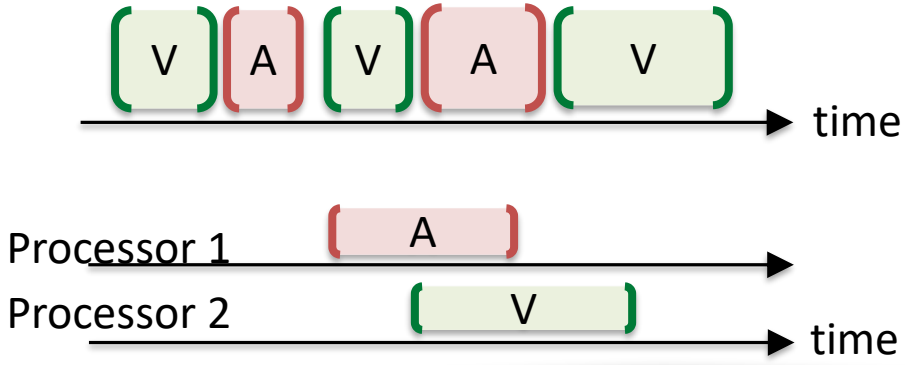
Anterior attacks must be performed ***before*** the **victim task**

Posterior attacks must be performed ***after*** the **victim task**

Pincer attacks must be performed ***before and after*** the **victim task**



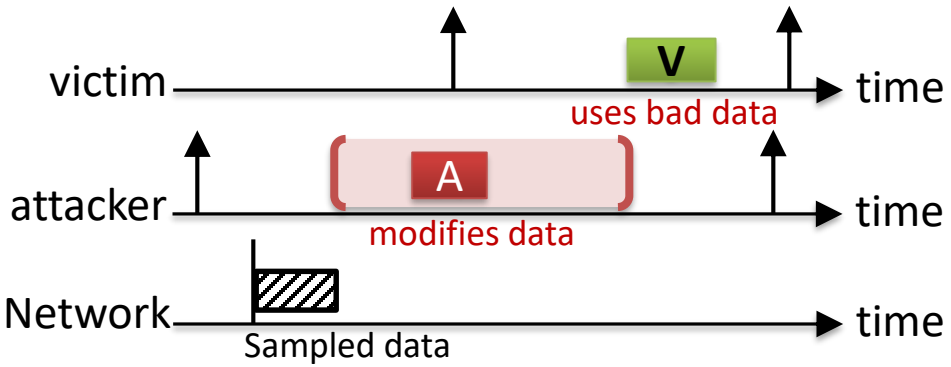
Concurrent attacks must be performed ***while the victim is running***

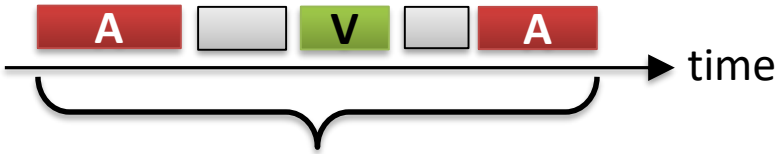


A taxonomy of schedule-based attacks

Anterior attacks | **Posterior attacks** | **Pincer attacks** | **Concurrent attacks**

Attacker's window depends on the goal of the attacker and the system specifications





Loose constraint

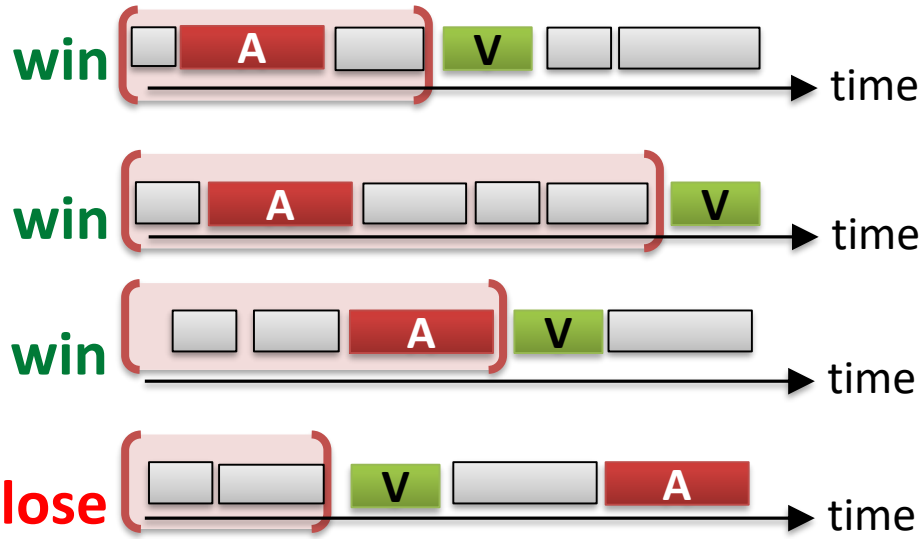
Tight constraint

Is schedule randomization a good defense
against schedule-based attacks?

It is a system-oblivious and attack-oblivious defense

A real-time system that uses cache partitioning to avoid CRPD, is strong against Pincer cache side-channel attacks, but it might be weak against data injection Anterior or Posterior attacks if it does not apply access control policies.

Consider an opportunistic anterior attacker:

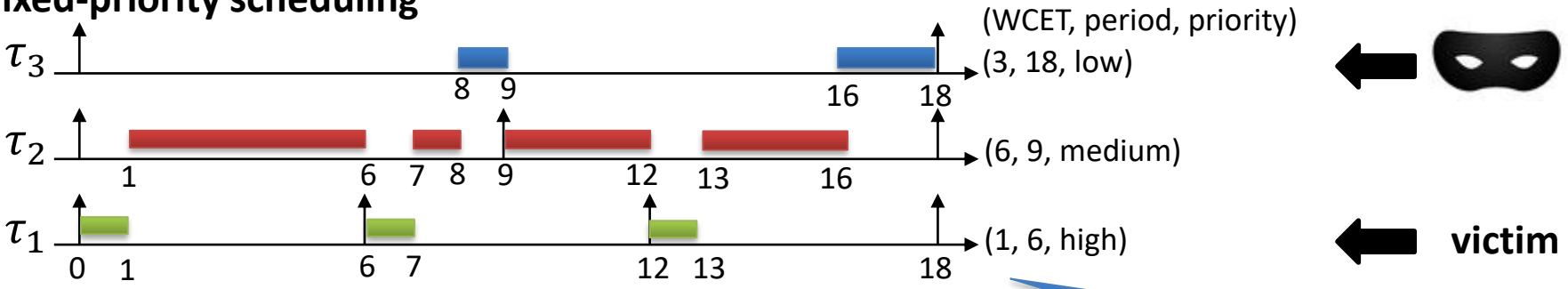


Is schedule randomization a good defense
against schedule-based attacks?

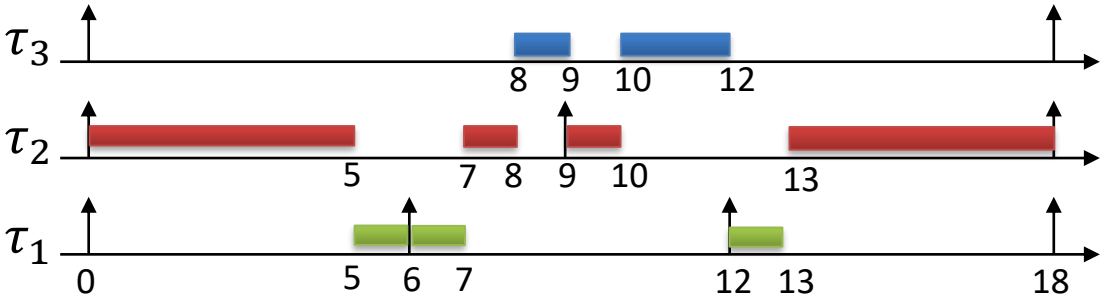
Opening Pandora's box

by **allowing** schedule-based attacks **that would have been impossible** when using a fixed-priority scheduling policy

Fixed-priority scheduling



Schedule randomization



τ_3 can never be directly scheduled before or after τ_1

τ_3 has a non-zero chance to achieve its goal

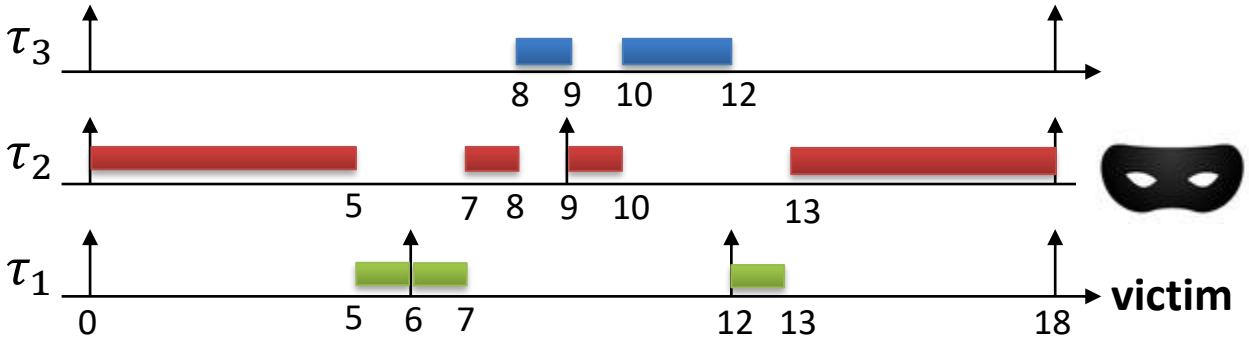
Is schedule randomization a good defense against schedule-based attacks?

Having **far less choices** (and **entropy**)
than other randomization-based defenses

Schedule randomization { **Time** × **Tasks**

Issue:
- These two dimensions **are related**
- Prior decisions will affect/limit the future decisions

Two box
Two balls



At time 5, τ_2 can **accurately guess** the schedule of τ_1

Is schedule randomization a good defense against schedule-based attacks?

There is **no valid security metric** to **evaluate** schedule randomization methods

- Existing metric is **schedule entropy** [Yoon et al., RTAS'16]: the uncertainty in the schedule of one hyperperiod (based on the Shannon entropy).

Is schedule randomization a good defense against schedule-based attacks?

There is **no valid security metric** to **evaluate** schedule randomization methods

Schedule entropy is not a security metric

Since it does not account for the attack (attack's goal, requirements, …)

It creates an **illusion of security**

Example:
The attacker τ_1 wants to be scheduled directly before τ_2 (victim)

Schedules

Time slots	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
1	τ_1	τ_1	τ_1	τ_3	τ_3	τ_3	τ_2	τ_2	τ_2	τ_2	τ_2	τ_2
2	τ_2	τ_2	τ_3	τ_1	τ_2	τ_2	τ_1	τ_1	τ_2	τ_2	τ_3	τ_3
3	τ_2	τ_3	τ_2	τ_2	τ_1	τ_2	τ_2	τ_3	τ_1	τ_3	τ_1	τ_2
4	τ_3	τ_2	τ_2	τ_2	τ_2	τ_1	τ_3	τ_2	τ_3	τ_1	τ_2	τ_1

All 12 random schedules that can be generated for three tasks $\tau_1 = (1, 4)$, $\tau_2 = (2, 4)$, and $\tau_3 = (1, 4)$.

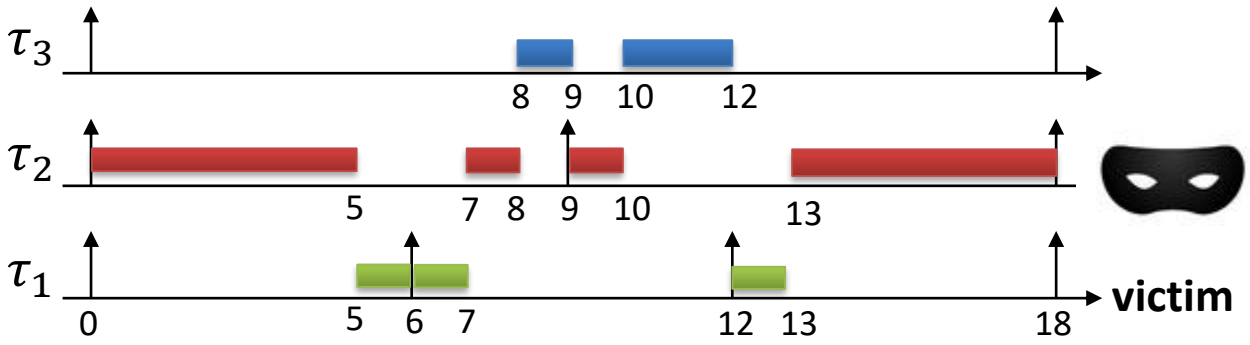
Schedule entropy = 3.58

Attack-aware entropy = 0.81

Is schedule randomization a good defense
against schedule-based attacks?

There is **no valid security metric** to **evaluate** schedule randomization methods

Schedule entropy is optimistic
since it does not account for the attacker's partial observations

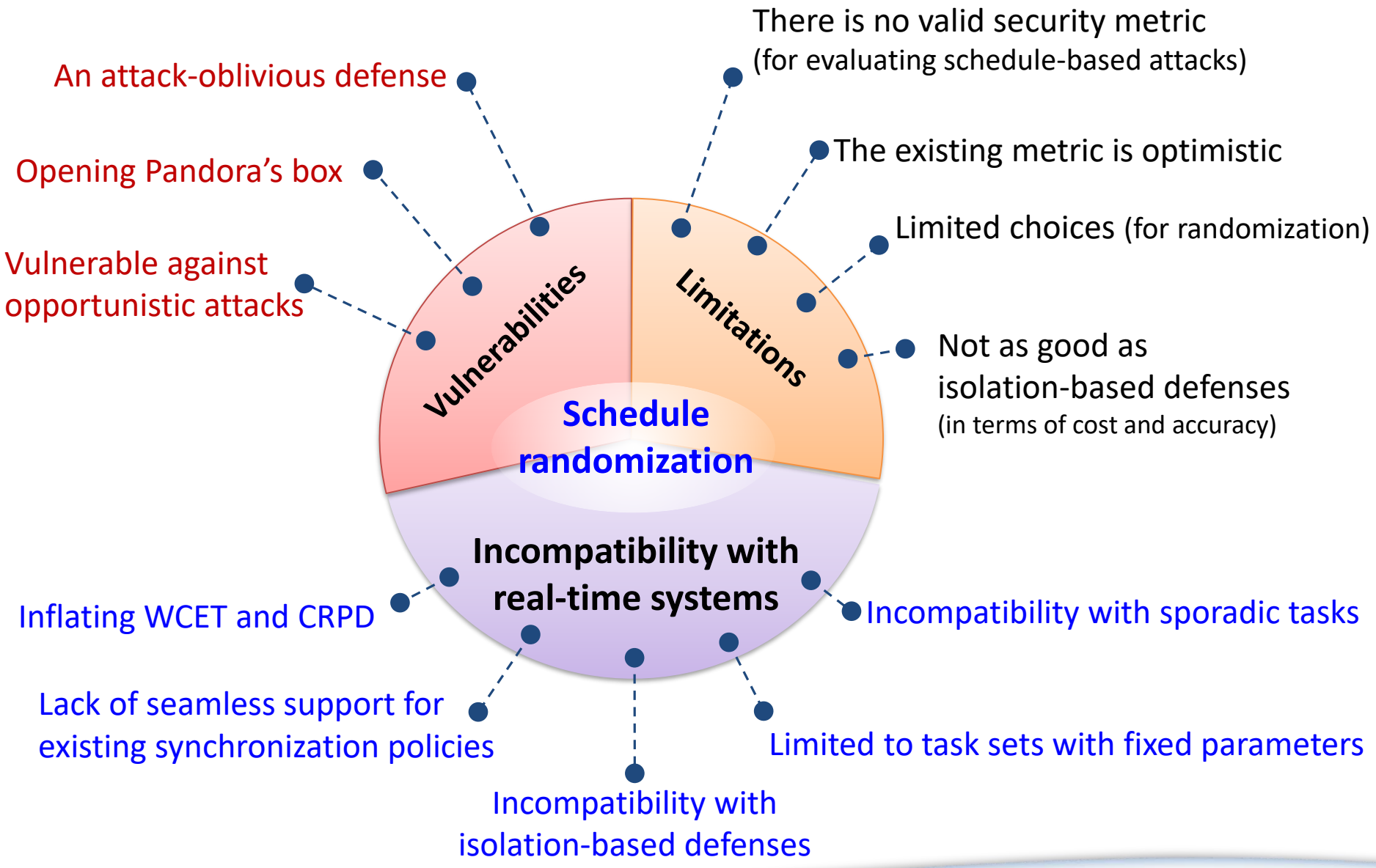


Schedule entropy = **18.96**

Conditional entropy of scheduling τ_1 at time 5 in this schedule = 0

Attacker's partial observations change the game of schedule uncertainty

Is schedule randomization a good defense against schedule-based attacks?



Agenda

- Taxonomy of schedule-based attacks
- Is schedule randomization a good defense against schedule-based attacks?

- **A preliminary security test for fixed-priority scheduling**

highlights

Given a task set with a set of **untrusted tasks**,
Derive a **set of conditions** to **determine** if an Anterior, Pincer, or Posterior attack can never happen in the system.

More in the paper

- **Evaluation**

(how well does the schedule randomization perform in various attack scenarios?)

Attack success ratio (ASR)
The chance that a victim job is (positively) attacked

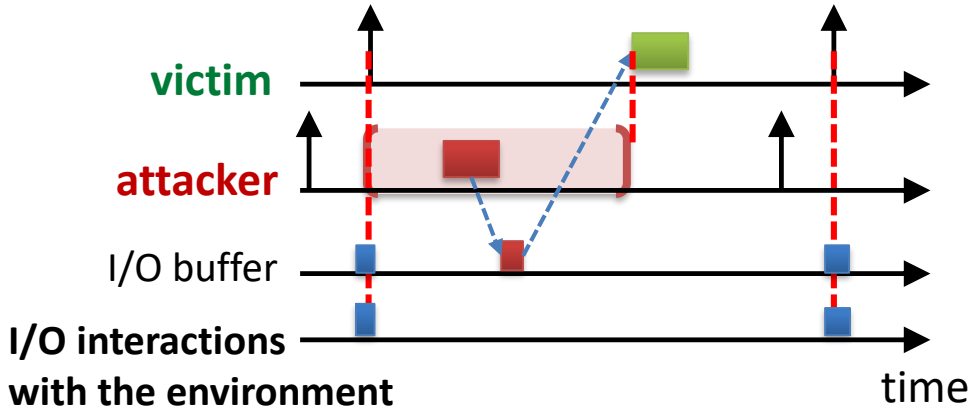
Rate Monotonic (Baseline)
TS1, TS2, TS3 (three versions of TaskShuffler)
Krueger (slot shifting)

How well does the schedule randomization perform in various attack scenarios?

System model

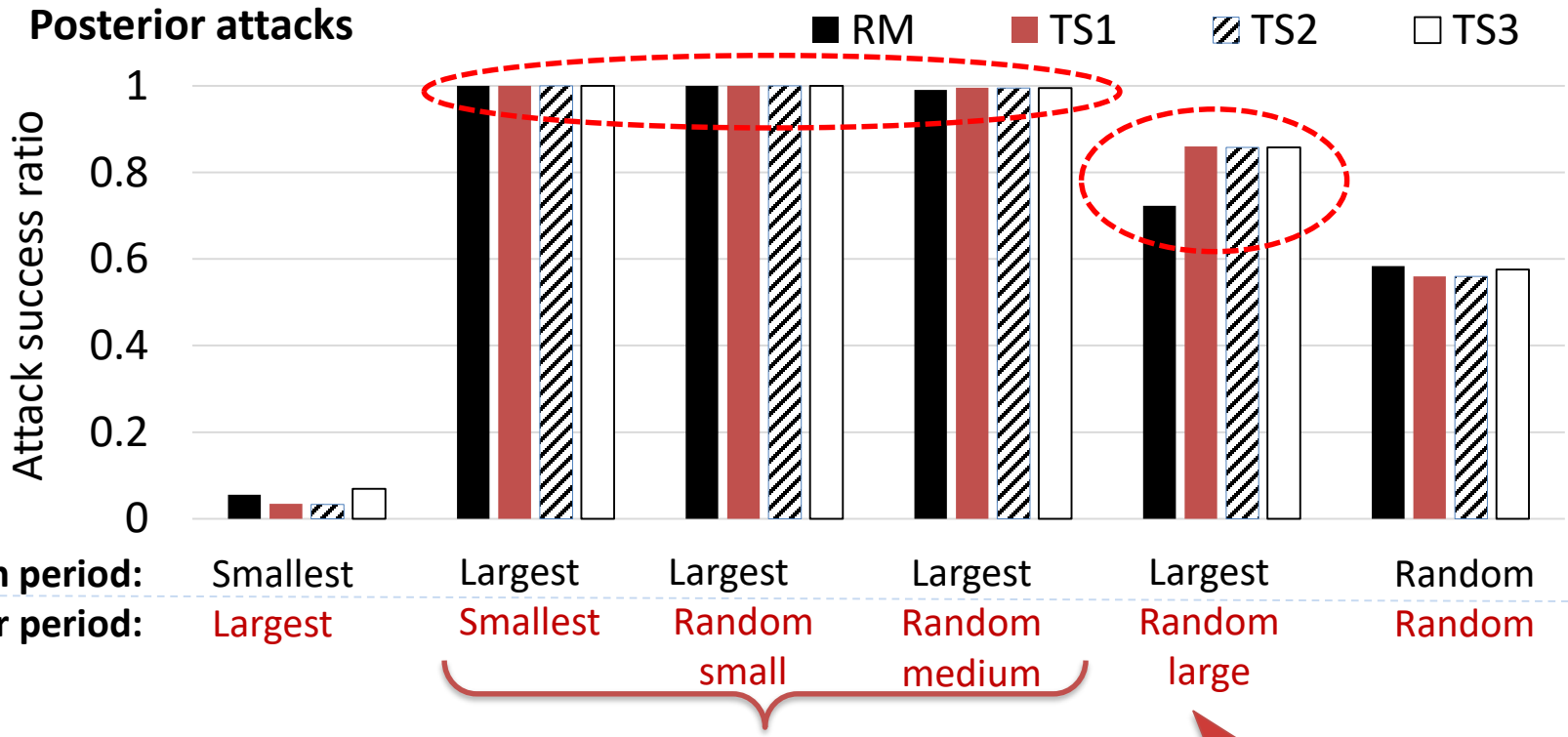
- Logical execution time (LET) architecture (Giotto [Emsoft'06])
  I/O interactions happen at the releases

Example: **Anterior attack**



Randomizing the schedule does not eliminate schedule-based attacks

More experiments in the paper



When attacker has a higher frequency of activation than the victim, it will eventually be able to execute "after" the victim.

Randomization is worse than baseline!

Priorities are assigned according to rate-monotonic priority assignment method
10 tasks, periods in {1, 2, 5, 10, 20, 50, 100, 200, 1000}ms, utilization in {0.1, 0.3, 0.5, 0.7}


WHAT'S NEXT?

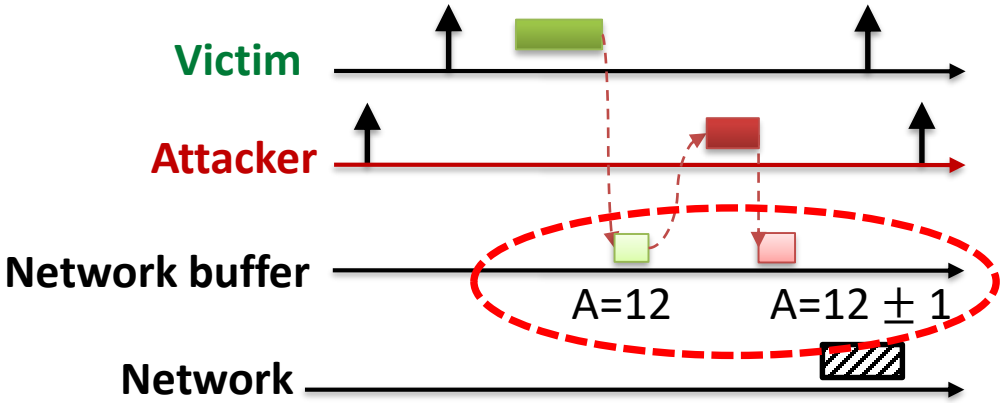
Future work

"questions"

"**Time predictability** makes real-time systems **vulnerable** against **schedule-based attacks**"

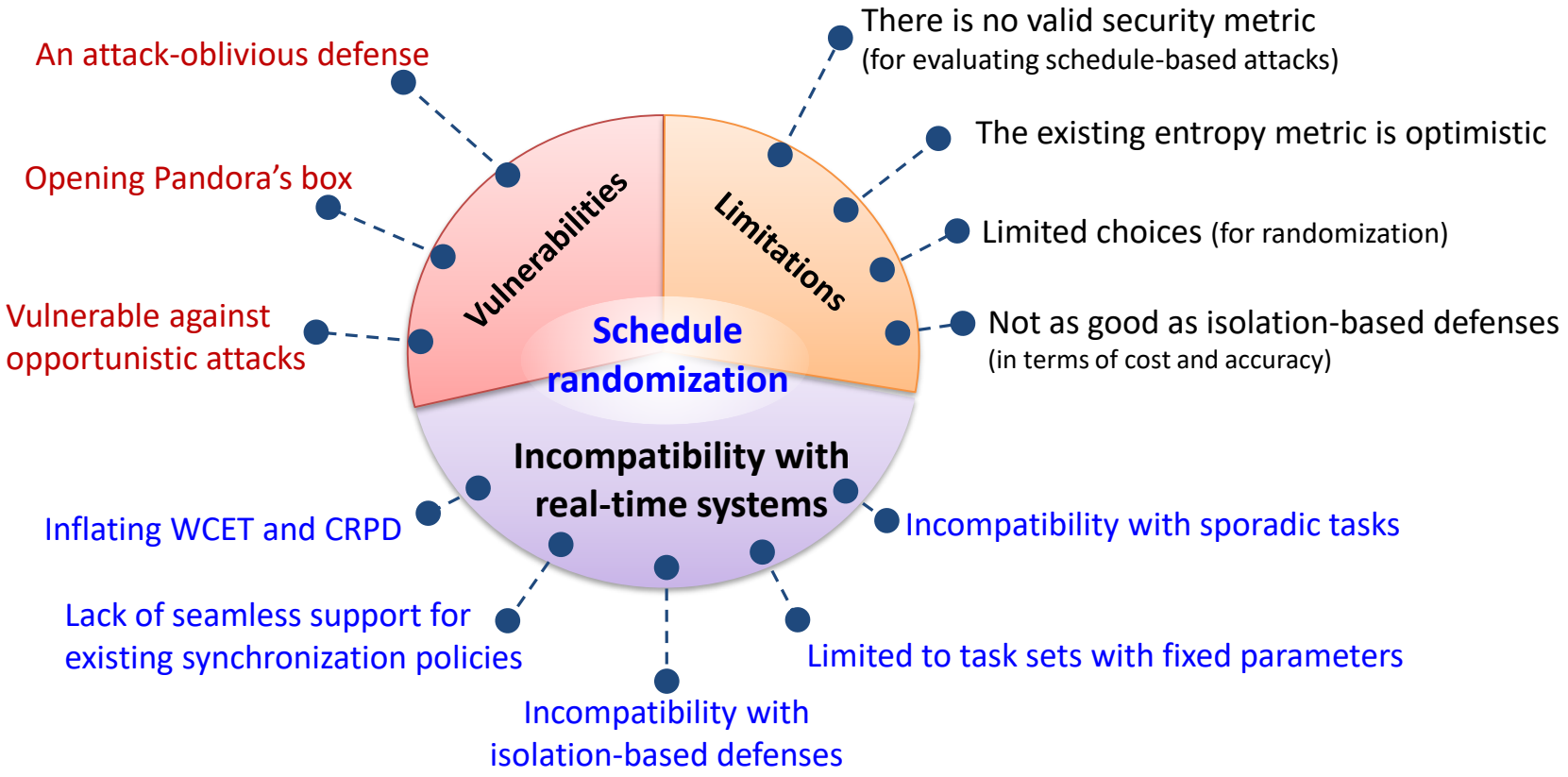
Do we really need to hide the schedule to defend the system against schedule-based attacks?

How can we implement **light-weight isolation-based defenses** for real-time embedded systems?



A simple access control policy could easily solve this security issue


Questions


An attack-oblivious defense
Opening Pandora's box
Vulnerable against opportunistic attacks
Vulnerabilities
There is no valid security metric (for evaluating schedule-based attacks)
The existing entropy metric is optimistic
Limited choices (for randomization)
Not as good as isolation-based defenses (in terms of cost and accuracy)
Limitations
Schedule randomization
Incompatibility with real-time systems
Inflating WCET and CRPD
Lack of seamless support for existing synchronization policies
Incompatibility with isolation-based defenses
Incompatibility with sporadic tasks
Limited to task sets with fixed parameters

Thank you