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

Mitra Nasri       Tam Chantem       Gedare Bloom       Ryan Gerdes
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.
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)

**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]

**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]

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.

---


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.

Victim task

Hijacked task (attacker)

Network buffer

Network

**A=30**

A=12

A=12 ± 1

A=30 ± 1

A=12

**Successful attack**

**Unsuccessful attack**
### A taxonomy of schedule-based attacks

<table>
<thead>
<tr>
<th>Anterior attacks</th>
<th>Posterior attacks</th>
<th>Pincer attacks</th>
<th>Concurrent attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anterior attacks must be performed <strong>before</strong> the victim task</td>
<td>Posterior attacks must be performed <strong>after</strong> the victim task</td>
<td>Pincer attacks must be performed <strong>before and after</strong> the victim task</td>
<td>Concurrent attacks must be performed <strong>while the victim is running</strong></td>
</tr>
</tbody>
</table>

**Anterior attacks**
- **Victim** task
- **Attacker** attacks

**Posterior attacks**
- **Victim** task
- **Attacker** attacks

**Pincer attacks**
- **Victim** task
- **Attacker** attacks

**Concurrent attacks**
- **Victim** task
- **Attacker** attacks

**Attacker’s net**

**Processor 1**
- **Attacker**

**Processor 2**
- **Victim**
Attacker’s window depends on the goal of the attacker and the system specifications.
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

\[ \tau_3 \]
\[ \tau_2 \]
\[ \tau_1 \]

Schedule randomization

\[ \tau_3 \]
\[ \tau_2 \]
\[ \tau_1 \]

\( \tau_3 \) can never be directly scheduled before or after \( \tau_1 \)

\( \tau_3 \) 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

---

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

---

**Schedule randomization**

\[
\begin{aligned}
\text{Time} & \times \text{Tasks} \\
\tau_3 & \quad 8 \quad 9 \quad 10 \quad 12 \\
\tau_2 & \quad 5 \quad 7 \quad 8 \quad 9 \quad 10 \quad 13 \\
\tau_1 & \quad 0 \quad 5 \quad 6 \quad 7 \quad 12 \quad 13 \quad 18 \\
\end{aligned}
\]

**At time 5, \( \tau_2 \) can accurately guess the schedule of \( \tau_1 \)**
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, ...)

Example:
The attacker $\tau_1$ wants to be scheduled directly before $\tau_2$ (victim).

<table>
<thead>
<tr>
<th>Time slots</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_9$</th>
<th>$S_{10}$</th>
<th>$S_{11}$</th>
<th>$S_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\tau_1$</td>
<td>$\tau_1$</td>
<td>$\tau_1$</td>
<td>$\tau_3$</td>
<td>$\tau_3$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
</tr>
<tr>
<td>2</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_3$</td>
<td>$\tau_1$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_1$</td>
<td>$\tau_2$</td>
<td>$\tau_3$</td>
<td>$\tau_3$</td>
<td>$\tau_3$</td>
<td>$\tau_3$</td>
</tr>
<tr>
<td>3</td>
<td>$\tau_2$</td>
<td>$\tau_3$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_1$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_3$</td>
<td>$\tau_1$</td>
<td>$\tau_3$</td>
<td>$\tau_1$</td>
<td>$\tau_2$</td>
</tr>
<tr>
<td>4</td>
<td>$\tau_3$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_2$</td>
<td>$\tau_1$</td>
<td>$\tau_3$</td>
<td>$\tau_2$</td>
<td>$\tau_3$</td>
<td>$\tau_1$</td>
<td>$\tau_2$</td>
<td>$\tau_1$</td>
</tr>
</tbody>
</table>

Schedule entropy $= 3.58$

Attack-aware entropy $= 0.81$

It creates an **illusion of security**

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

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 $\tau_1$ 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?

Vulnerabilities

- An attack-oblivious defense
- Opening Pandora’s box
- Vulnerable against opportunistic attacks

Limitations

- Not as good as isolation-based defenses (in terms of cost and accuracy)
- Limited choices (for randomization)
- The existing metric is optimistic

Vulnerabilities vs. Limitations

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
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.

**Evaluation**

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

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)
Randomizing the schedule does not eliminate schedule-based attacks

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.
An attack-obliscious defense

Opening Pandora’s box

Vulnerable against opportunistic attacks

Inflating WCET and CRPD

Lack of seamless support for existing synchronization policies

Incompatibility with isolation-based defenses

There is no valid security metric (for evaluating schedule-based attacks)

Not as good as isolation-based defenses (in terms of cost and accuracy)

The existing entropy metric is optimistic

Limited choices (for randomization)

Incompatibility with sporadic tasks

Limited to task sets with fixed parameters

Incompatibility with real-time systems

Schedule randomization