Real-Time Systems Security through Scheduler Constraints

SIBIN MOHAN, MAN-KI YOON, RODOLFO PELLIZZONI AND RAKESH BOBBA

Information Trust Institute
Dept. of Computer Science

Dept. of Electrical and Computer Engineering

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNIVERSITY OF WATERLOO
Introduction

- Real-time systems (RTS) considered to be invulnerable to external security attacks
  - Due to use of proprietary hardware/protocols
  - Physical isolation
- Above assumptions are being challenged
  - Subsystems interconnected with each other (even through the Internet)
  - Malware developers able to overcome air gaps
  - Attacks demonstrated on automobiles, avionics systems, UAVs, power grids, etc.
- Security violations in real-time systems could be more catastrophic than other systems
  - Loss of life, physical harm to humans, system & environment, etc.
- Cannot tack on regular security mechanisms without concern for real-time properties
Contributions

Problem: Information leakage in real-time systems
- Use of shared resources (e.g.: caches, DRAMs, etc.) to leak critical data
- Between tasks with different security levels

Contribution: integrate security at design phase of RTS using intelligent scheduling constraints

- Fixed-priority (FP) scheduling schemes
- Analysis bounds for the integration of such constraints in FP algorithms
Outline

- System model, adversary model, assumptions
- Security problem, Outline of our Solution
- Scheduling constraints: PreFlush, Half-PF, Constrained PreFlush
- Analysis
- Further scheduling considerations: Ordering
- Evaluation
- Conclusion
Assumptions, adversary model, etc.

- Information leakage possible in systems with *multiple levels of security*
  - E.g.: DO-178B style avionics system with navigation system (low) and flight control (high)

- Security levels could *differ* from real-time priorities
  - E.g.: UAV with camera and real-time control tasks
  - Image capture and processing tasks → higher requirements for confidentiality
  - Real-time control tasks (flight path, engine control, etc.) → higher real-time priorities

- **Adversary**
  - Can insert new tasks or compromise existing tasks → respects RT guarantees to avoid detection
  - Passively gleans secure information → by observation of shared resource usage
  - *Cannot* observe RAM contents of other tasks
  - *Cannot* tamper with system operation
System Model

- Real-Time
  - Liu and Layland task model
  - Set of sporadic tasks
  - Fixed-priority (FP) scheduling algorithm

- Security
  - Set of security ‘levels’ of tasks forms a total order
  - Given any two tasks, $\tau_i$ and $\tau_j$, ‘security ordering’ can be one of
    $S_i \leq S_j$ or $S_j \leq S_i$
  - Will generalize to a partial order in future work
Security problem

- Information leakage through storage channels over implicitly shared resources

- Consider two tasks, $H$ and $L$ such that
  - $H$ has higher real-time priority than $L$
  - $H$ is also at a higher security level than $L$
  - Hence, $L$ should not be privy to $H$’s information/internal state

- If $L$ follows $H$ at any point, then there is potential that a compromised $L$ can snoop upon $H$

- Consider:
Solution

1. **Clean up the shared resource** (eliminating storage channel)  
   - Between every transition from/to H and L  
   - E.g.: flush the cache after each task has completed

2. **Scheduling constraints** to prevent situations where leakage can occur  
   - No instance of L can be scheduled after any instance of H  
   - If an instance of L is preempted by H and then resumes later, leakage can still occur → avoid

   From an implementation perspective, the above constraints translate to:
   a. Flush/clean out shared resource on every transition of type H → L  
   b. Ensure that all jobs of H complete before transitioning to L  
   c. Prevent L from being preempted by H once it has started executing
**PreFlush (PF), Half-PF**

- Rules for PF are
  1. For every pair of tasks, $\tau_i$ and $\tau_j$, such that $\tau_i \rightarrow \tau_j$, invoke $\text{FT}$ on every transition, $\tau_i \rightarrow \tau_j$.
  2. Invoke $\text{FT}$ on every transition of type, $\tau_j \rightarrow \tau_i$.

- Second rule prevents ‘responses’, i.e. confirmations in a covert channel setup.
- If the first rule is active then even if the responses can be sent, it doesn’t matter.

**HALF-PF**

![Diagram of Flush Task (FT) and half-PF states](image-url)
Constrained PreFlush (CPF)

- Rules for CPF are
  1. For every pair of tasks, $\tau_i$ and $\tau_j$, such that prevent $\tau_i$ from preemption of $\tau_j$
  2. For every pair of tasks, $\tau_i$ and $\tau_j$, such that allow $\tau_i$ to preempt $\tau_j$

- For the first rule, if there exist one or more tasks such that
  - $\tau_i$ is still allowed to execute after $\tau_j$ $\Rightarrow$ avoids situation where $\tau_i$ faces inordinate priority inversion
  - We are concerned more with direct priority inversion and not indirect ones
FP and Security

- Fixed Priority (FP) Schedulers are a class of well known static scheduling algorithms
- We show how to integrate the Half-PF constraint into FP scheduling algorithms
  - Start with non-preemptive FP schedulers → one of the easier algorithms to implement/analyze
- Our techniques
  1. Provide insights into how security-related constraints can be integrated into scheduling algorithms
  2. Demonstrate how worst-case response-time analysis can be carried out for such situations

Let
- \( \tau_i \): task under analysis
- \( c_{ft} \): execution time for one invocation of the flush task (FT)
- FTs are executed non-preemptively
Analysis

- Analysis strategy
  - Use standard response-time analysis for non-preemptive FP
  - Compute number of higher or equal priority jobs that interfere with $\tau_i$
  - Determine maximum number of FT invocations required by such jobs $\implies$ increase response times
  - Iterate until convergence is achieved

- Worst-case response time of task $\tau_i$ at iteration ‘$k$’,

$$R_i(k+1) = B_i + N_{ft}(S, \{I_j | \tau_j \in hep_i\})c_{ft} + \sum_{\forall j \in hep_i} (I_jc_j) + c_i$$

$I_j$: number of instances of higher or equal priority task $\tau_j$ that interfere with $\tau_i$

$$I_j = \left\lfloor \frac{R_i(k) - c_i}{p_j} + 1 \right\rfloor$$

$B_i$: max. blocking time

$$B_i = \max_{\forall \tau_j \in l_{p_i}} \bar{c}_j - 1$$
\[ R_i(k+1) = B_i + N_{ft}(S, \{I_j | \tau_j \in hep_i\})c_{ft} + \sum_{\forall j \in hep_i} (I_j c_j) + c_i \]

- \( N_{ft} \): worst-case number of FT required by interfering higher/equal priority tasks

- \( N_{ft} \) derived only using
  - Ordering of security levels
  - Number of interfering jobs that are of higher or equal priority
  - No assumptions on arrival times or other parameters of higher/equal priority jobs

- In the paper demonstrate how to compute \( N_{ft} \) in polynomial time in the number of jobs
Analysis (contd.)

- $N_{ft}$ computation: base idea
  - Create a flow graph where nodes represent jobs and edges represent FT
  - Each job is represented by a “sender” and a “receiver” node
  - $\text{SendF}$ represents any job executed before the busy interval; $\text{RecvL}$ is the job under analysis
  - Run max flow algorithm

$S_1 \rightarrow S_2 \rightarrow S_3$

$\tau_3$ under analysis

$l_1 = 1$

$l_2 = 2$
Further Scheduling Considerations

Important issues arise when trying to integrate security into RT systems

1. What is the best ordering of security levels?
2. Is there such a thing as the “best” ordering of security levels?
3. If it exists, is this “best” ordering in any way related to the real-time priorities of the system?

Answer: depends!

- Can provide some hints to designers

- **Forward Ordering:** For every pair of tasks, $\tau_i$ and $\tau_j$

- **Backward Ordering:** For every pair of tasks, $\tau_i$ and $\tau_j$

- **Random Ordering:** no real relationship between task priorities and security levels
## Ordering & Constraints

<table>
<thead>
<tr>
<th></th>
<th>Forward Ordering</th>
<th>Backward Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Half-PF</strong></td>
<td>Every transition of type $\tau_i \rightarrow \tau_j$</td>
<td>Least number of FT invocations</td>
</tr>
<tr>
<td></td>
<td>• $\tau_i$ has higher priority than $\tau_j$</td>
<td>• Transition from higher to lower priority $\rightarrow$ transition from lower to higher security levels</td>
</tr>
<tr>
<td></td>
<td>• will result in an FT invocation</td>
<td>• Execute FT at preemptions only</td>
</tr>
<tr>
<td></td>
<td>• chances are high that most preemptions will result in FT</td>
<td></td>
</tr>
<tr>
<td><strong>CPF</strong></td>
<td>Prevents preemptions, but still suffers from overheads of many FT invocations</td>
<td>Same as Half-PF – all preemptions are by lower security tasks</td>
</tr>
</tbody>
</table>
Evaluation

- Set up simulation and analysis engines
- Generated and analyzed 2000 synthetic task sets
  - 10 base utilization groups: \([0.02+0.1xi, 0.08+0.1xi]\) for \(i = 0 \ldots 9\)
    - [base utilization: total utilization for the tasks in set]
  - Task parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks, (N)</td>
<td>[3, 10]</td>
</tr>
<tr>
<td>Task periods, (p_i)</td>
<td>[50, 100, 150, … 950, 1000]</td>
</tr>
<tr>
<td>Task execution times, (e_i)</td>
<td>[3, 30]</td>
</tr>
<tr>
<td>FT overhead</td>
<td>{1, 5, 10}</td>
</tr>
</tbody>
</table>

- Task deadlines = periods
  - assigned priorities based on Rate Monotonic (RM) algorithm

Used same task sets for both
1. Evaluation of analysis bounds
2. Simulation-based evaluation


[2] executes task sets up to one hyperperiod; system tracks response times for each task.

On completion of a job, both check whether response times exceed task deadlines.
Analysis-based Results

Non-Preemptive FP with Random ordering, Half-PF constraint

- Vanilla FP performs best $\rightarrow$ no constraints
- Our method [black line] $\rightarrow$ better than naïve bounds for number of FT invocations
- Designers can see effects of security constraints
- Reduced schedulability, but increased security

$\text{FT} = 5$
Analysis-based Results (contd.)

Varying FT Overhead Costs

As FT overheads go up, our analysis-based methods perform better \(\rightarrow\) compared to naïve bounds.
As FT overheads go up, preemptive FP performs worse \(\rightarrow\) more FT compared to non-preemptive.
Simulation-based Results

- Use a simulator that schedules task sets according to one of the following:
  1. Preemptive (vanilla) FP: preemptions allowed, no FT invocations \([\text{FP}]\)
  2. NonPreemptive FP: FP no preemptions; FT allowed between high \(\rightarrow\) low security level transitions \([\text{FP FULLY NON-PREEMPTIVE}]\)
  3. Preemptive FP with flush tasks: FT invoked on transitions from high \(\rightarrow\) low; Half-PF constraint \([\text{FP HALF-PF}]\)
  4. Preemptive FP with resource flush under certain conditions \([\text{FP CPF}]\)

- FT overhead was set to 5 for all simulation experiments
Simulation-based Results (contd.)

Random Security Ordering, FT = 5

- Vanilla FP performs best → no constraints
- FP FULLY NON-PREEMPTIVE → worst
  [no preemptions at all]
- FP HALF-PF and FP CPF perform much better
- Both start dropping off around 75% utilization

Real-Time Systems Security Through Scheduler Constraints
Ordering + Simulation Results

**Forward Security Ordering, FT = 5**

Performs the worst → more FT invocations

**Backward Security Ordering, FT = 5**

Performs the best → least FT invocations
FT: Simulation vs Analysis

- Number of FT invocations normalized to number of jobs
- Red dots: FT invocations [simulation]
- Blue dots: FT invocations [analysis]

Hence

1. Num. FT invocations much less than number of jobs
2. For most task sets, number of actual FT invocations lower than calculated values
3. True even for higher utilization task sets!
Limitations

- Transforming security requirements into scheduling constraints
  - Our solution for one problem \(\rightarrow\) information leakage through storage channels
  - Not a silver bullet for all security problems in real-time systems

- Many security properties may not be amenable to being cast as scheduling constraints
  - E.g.: communication-related vulnerabilities

- Performance overheads could inhibit adoption in many RTS
  - May be mitigated by careful design process
Conclusion

- Presented methods to integrate security properties into real-time systems
- Techniques to amend FP algorithms to reduce information leakage through shared resources
- Designers of real-time systems can now consider such security properties
  - Can assess tradeoffs between security requirements and real-time guarantees
- Future Work
  - Analysis for other scheduling policies / constraints
  - Case study
  - Architectural mechanisms?
Thanks!

- Questions?