T-Tex: Timed Threaded Execution for Real-time Security and Safety (OpenMP)

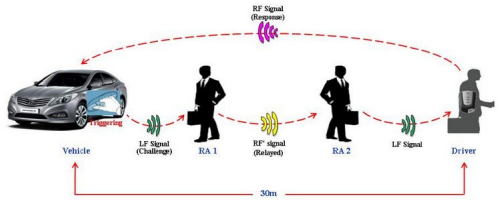
Swastik Mittal, Frank Mueller

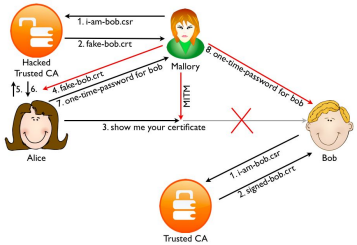
Contents

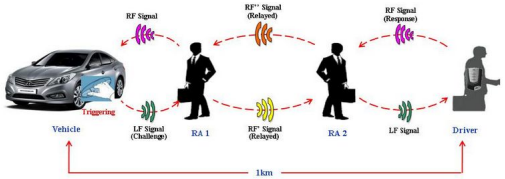
Real Time Systems & Security

- **Predictable systems**
  - **Predictability is key over performance**
  - **Event triggered real time systems**
    - **E.g.:** Braking system in autonomous vehicles
  - **Time triggered real time systems**
    - **E.g.:** Subsystem within drones
      - Drone: periodically communicate navigation and flight parameters.

- Process of detecting and preventing intrusion
- Two broad categories:
  - **System Security**
  - **Network Security**

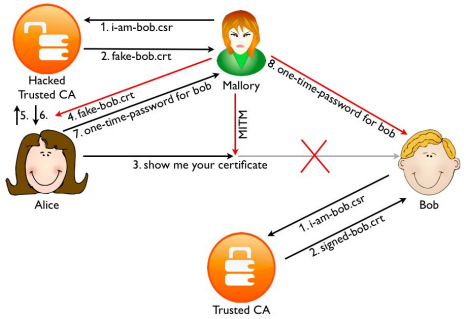


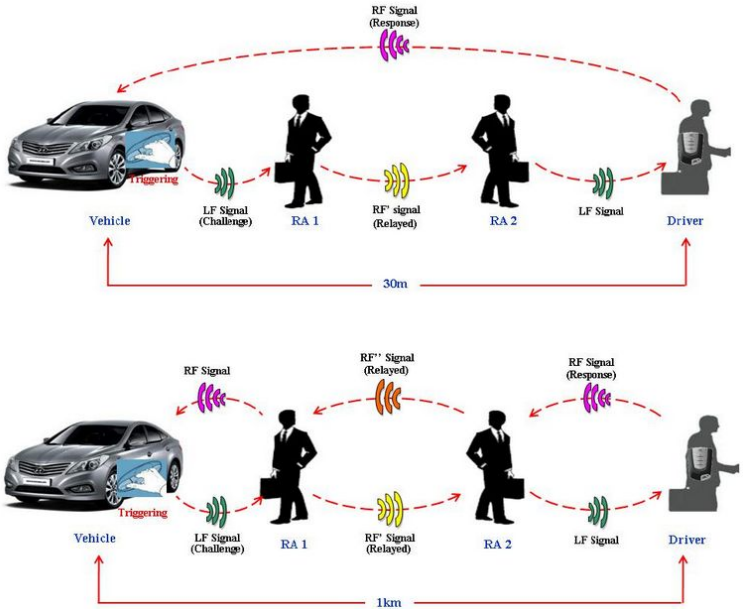




Computer Security

- Process of detecting and preventing intrusion
- Two broad categories:
  - **System Security**
  - **Network Security**



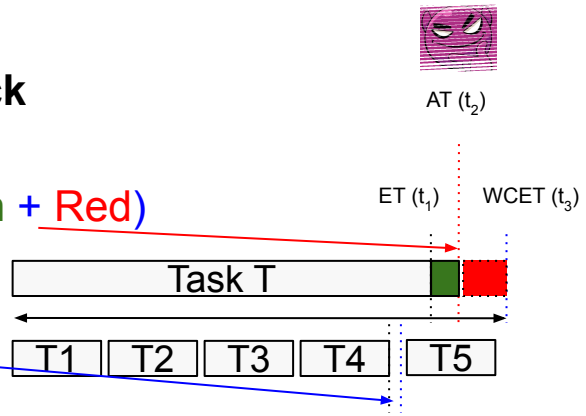


Attacks

- **System Attacks**
  - Buffer overflow
  - SQL Injection
  - Privilege escalation etc.
- Network Attacks
  - Man-In-the-Middle
    - E.g.: Relay and Replay attacks. (Common attacks in real time systems)
  - Denial of service
    - E.g.: ping-of-death, syn-flooding etc.
- **Delay Attacks**
  - Delay in intended execution time of the task/process
  - Fatal in a real-time system
    - Car crashing because of break delay

Takeaway

- Attacker may not take over system
  - But can **easily** cause delays (deadline miss)
  - **Need to detect such delays as early as we can**
- Predictable nature of real-time systems enable execution-time analysis
  - **Worst case execution time of a task (WCET)**
  - **For an attacker to execute an undetected delay attack**
    - mask delay using slack time
    - security vulnerability threshold (WCET - ET : Green + Red)
  - **Finer division of such tasks → lower WCET**
    - lower vulnerability (Eg. T4)
    - Early detection before significant impact
  - **Higher observed execution time compared to expected time (WCET) → signalled as delay attack.**



OpenMP

- **Multiprocessing framework**
  - widely used paradigm in parallel programming
- **Task scheduling** in OpenMP
  - **Implicit**: fine-grained data/loop parallelism (E.g., #pragma omp for)
  - **Explicit**: fine-grained parallelism originally for recursion (E.g., #pragma omp task)
    - Used by real-time community as coarse-grained tasks
      → abstraction inversion, mismatch (see results in papers)
- **OpenMP lacks real-time capabilities**
- **OpenMP-RT [LCTES'24]** fills this gap: introduces real-time capabilities to OpenMP
  - Priority based thread pools
    - Schedule OpenMP threads utilizing linux real-time priority band
  - Supporting access to shared resources
    - in a lock free and wait free manner
- **Execution time analysis of such real-time systems is critical**
  - **Early intrusion detection → our T-Tex work (coming next!)**

OpenMP

- **Multithreading framework**
  - widely used paradigm in parallel programming
- **Task scheduling** in OpenMP
  - **Implicit**: fine-grained data/loop parallelism (E.g., #pragma omp for)
  - **Explicit**: fine-grained parallelism originally for recursion (E.g., #pragma omp task)
    - Used by real-time community as coarse-grained tasks
- **OpenMP lacks real-time capabilities**
- **OpenMP-RT [LCTES'24]** fills this gap: introduces real-time capabilities to OpenMP
  - Priority based thread pools
    - Schedule OpenMP threads utilizing linux real-time priority band
  - Supporting access to shared resources
    - in a lock free and wait free manner
- **Execution time analysis of such real-time systems is critical**
  - **Early intrusion detection → our T-Tex work (coming next!)**

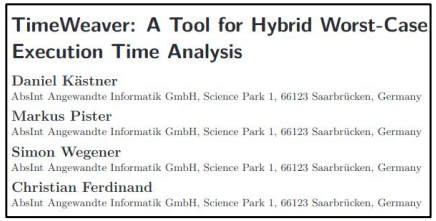
Contents

- Introduction
  - Real-Time Systems
  - Computer Security
  - OpenMP
- T-Tex
  - Related Work (WCET Analysis, Delay Detection)
  - Assumptions & Attack Model
  - Idea & Design
  - Implementation
  - Experimental Framework & Evaluation
  - Conclusion

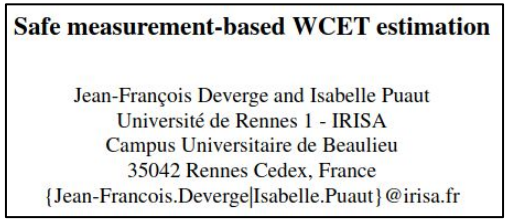
T-Tex Novelty

- Code execution time analysis is not new
- T-SYS [ICCPS'22]
  - First to do complete kernel code protection via execution time analysis
    - but is **single threaded**

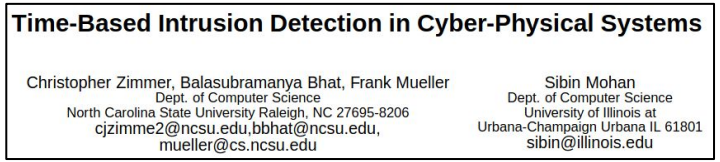
- **T-Tex contributions & novelty:**
  - **kernel timer crediting (never done before)**
    - Accurate execution time analysis
      - Eliminate blocking time by other task
  - **finer code region than basic block**
    - Instruction level
  - **Protecting all forms of loops**
    - statically or dynamically bounded


TimeWeaver: A Tool for Hybrid Worst-Case Execution Time Analysis

Daniel Kästner
AbsInt Angewandte Informatik GmbH, Science Park 1, 66123 Saarbrücken, Germany
Markus Pister
AbsInt Angewandte Informatik GmbH, Science Park 1, 66123 Saarbrücken, Germany
Simon Wegener
AbsInt Angewandte Informatik GmbH, Science Park 1, 66123 Saarbrücken, Germany
Christian Ferdinand
AbsInt Angewandte Informatik GmbH, Science Park 1, 66123 Saarbrücken, Germany

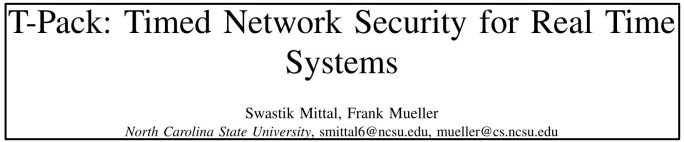

Safe measurement-based WCET estimation

Jean-François Deverge and Isabelle Puaut
Université de Rennes 1 - IRISA
Campus Universitaire de Beaulieu
35042 Rennes Cedex, France
{Jean-Francois.Deverge|Isabelle.Puaut}@irisa.fr


Time-Based Intrusion Detection in Cyber-Physical Systems

Christopher Zimmer, Balasubramanya Bhat, Frank Mueller
Dept. of Computer Science
North Carolina State University Raleigh, NC 27695-8206
cjzimme2@ncsu.edu,bbhat@ncsu.edu,
mueller@cs.ncsu.edu

Sibin Mohan
Dept. of Computer Science
University of Illinois at
Urbana-Champaign Urbana IL 61801
sibin@illinois.edu


T-Pack: Timed Network Security for Real Time Systems

Swastik Mittal, Frank Mueller
North Carolina State University, smittal6@ncsu.edu, mueller@cs.ncsu.edu

Motivation & Hypothesis

- Motivation
  - **Early detection** of delay
    - **no cost**
    - minor performance overhead for multithreaded RTS
      - uses simple monitoring techniques
- Hypothesis
  - Finer code division of OpenMP application
    - WCET analysis
  - Accurate execution time by considering overheads
    - Context switching
    - Resource contention

Contents

Assumption & Attack Model

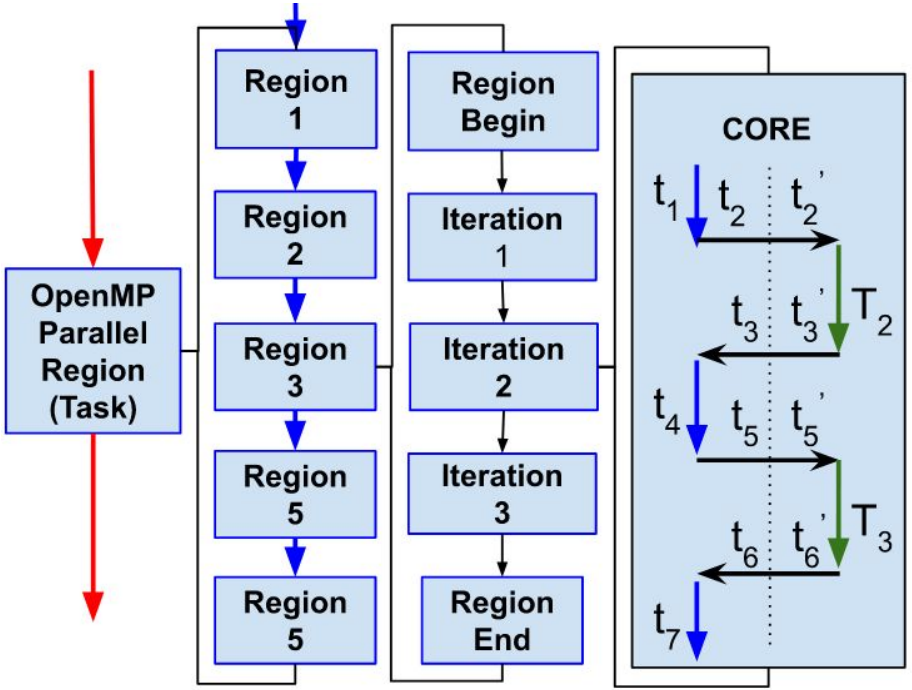
- **Oversubscription of threads on cores:** Reap parallel programming performance benefits
- **Core pinning:** Unbounded delays because of thread migration
- **Maximum number of parallel executing tasks known a priori**
  - Critical for static WCET analysis
- **Attack Scenarios**
  - **Kernel protected from any attack** (task priority & timer modification is not possible)
  - **Number of threads executing code region cannot be modified during execution**
    - Workload distribution already done by OpenMP
  - Possible attacks
    - **Buffer overflow attacks**
    - **Scheduling additional tasks** (not affecting the number of context switches)
      - Shared resources delay (memory delays)

Contents

Idea

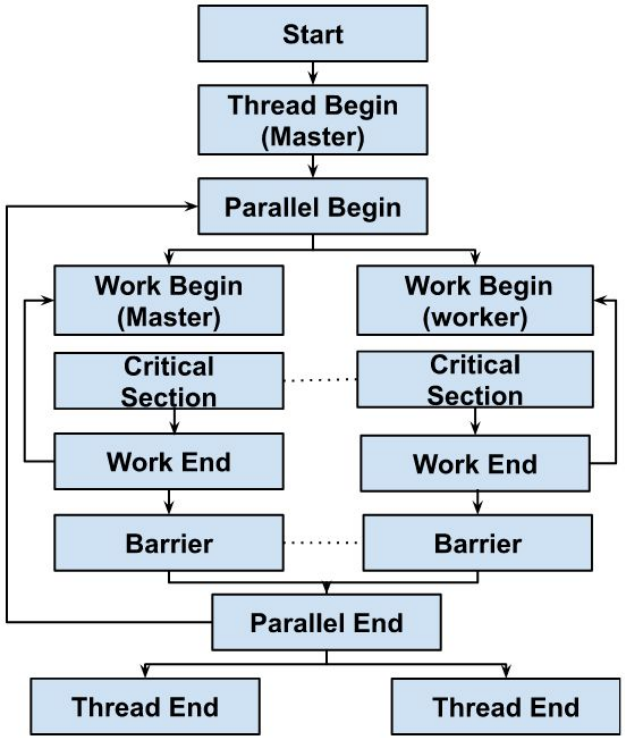
- **T-Tex: Threaded Time Execution tracking**
  - per-thread time varies depending on core contention
- **Identifying OpenMP regions**
  - Each region can be executed by different # threads
    - #pragma omp for vs #pragma omp sections
    - Affecting execution time of the region
- **Loop Identification**
    - Execution time of loops: depends on # threads
    - Timing loops is critical
      → need to identify static and dynamically bounded loops
- **Kernel timer crediting (Novel execution time analysis)**

Design: Protected Region



- Code division
  - Parallel region → multiple regions 1-5
- Region division
  - Each region further divided
    - Eg: Loop (3 code regions)
    - Loop of 30: (0-10,10-20,20-30)
- Thread suspension accounting
  - Eliminate T_2 & T_3 for accurate execution time
  - Blocks timed task

Identify OpenMP Regions (OMPT)

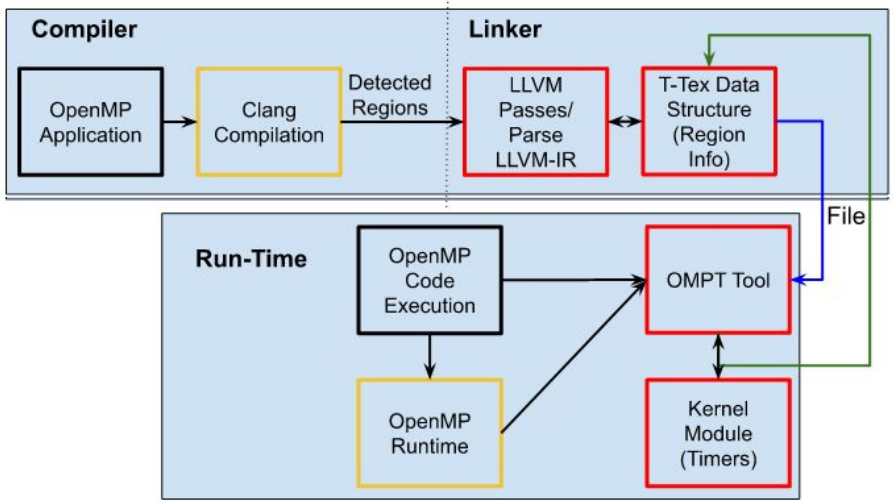


OpenMP Control Flow Graph

- Parallel Region (parallel Begin)
  - **implicit task**
  - **executing at different thread priority**
    - See OpenMP-RT
- **OMPT (OpenMP Tracer interface)**
  - Enables run-time profiling
  - **Identifies OpenMP regions using callbacks** at run-time
    - Sub regions using work begin
    - How to identify specific regions?
      - Callbacks **modified to pass id values**
  - **Maintains unique thread specific data → as a void pointer**
    - Store per thread timer information

Contents

Implementation: LLVM+Clang

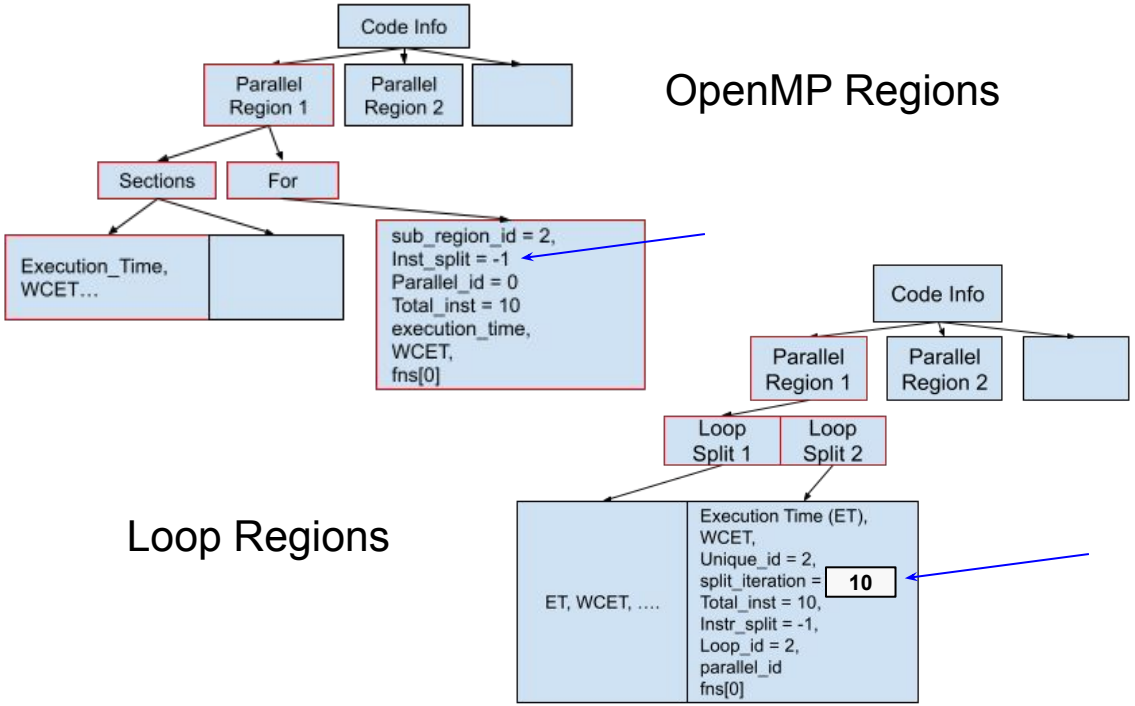


T-Tex Execution Pipeline: LLVM Compiler tool chain used for code instrumentation

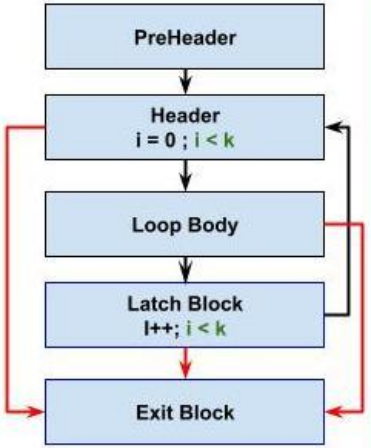
- **Clang parser**: Regions detected via AST
- **LLVM Pass in Linker**: Supports multiple files
- **Phase 0**: Pessimistic approach, identifies all regions
- **Region Handling**:
  - OMPT pre-identifies regions
  - Runtime calls modified to pass IDs
- **Loop Tracking**: Detects first loop iteration, stores ID-related info
  - **Attaches custom added callback (in OMPT)**
- Save region information within T-Tex Data Structure
  - Store details in a file

T-Tex Data Structure

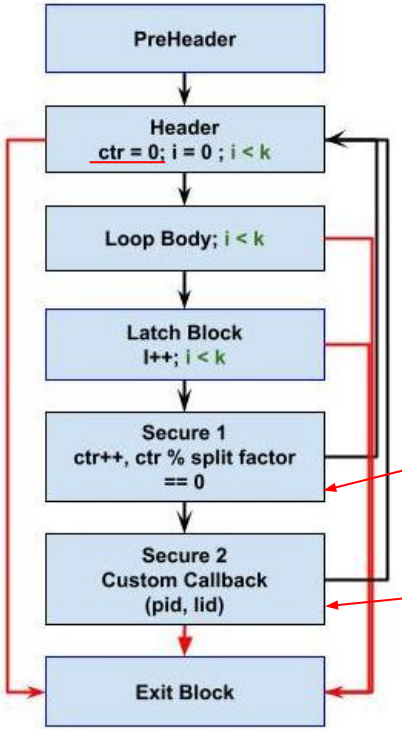
```
#include <bits/stdc++.h>

Tabnine | Edit | Test | Explain | Document | Ask
void func(){
}

Tabnine | Edit | Test | Explain | Document | Ask
int main(){
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                func();
            }

            #pragma omp section
            {
                printf("test");
            }
        }

        #pragma omp for
        {
            for(int i = 0 ; i < 30 ; i++) {
                for(int j = 0 ; j < 100 ; j++) {
                    func();
                }
            }
        }
    }
}
```



How T-Tex Splits Loops



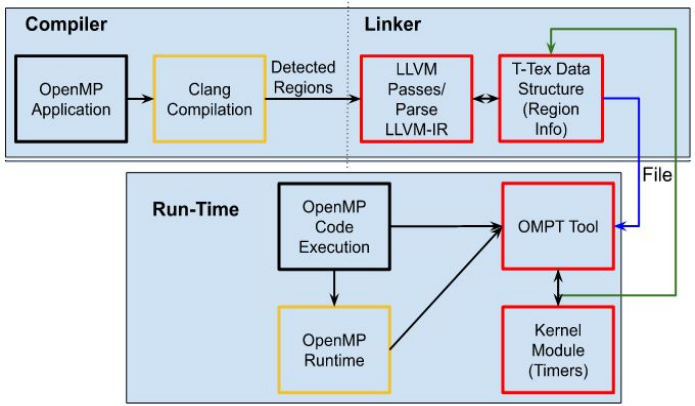
a)
Loop CFG without T-Tex



b)
Loop CFG with T-Tex

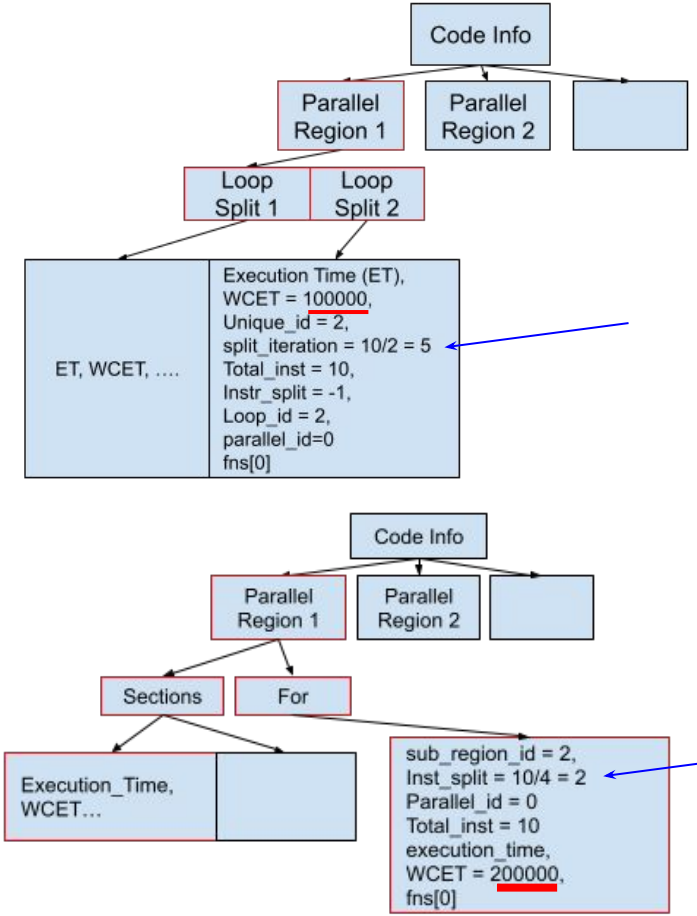
- **Identify statically & dynamically bounded loops**
  - difficult to track induction variable; instead:
  - **Add additional induction variable (ctr)**
    - LLVM phi-node
      - ctr value in header depends on calling block
      - 0 for preheader
- Add additional basic blocks
- Secure1
  - Checks & increments
  - Call header or Secure2
- Handles multiple scenarios
  - LatchBlock != Header
  - LatchBlock == Header
  - Multiple LatchBlocks

At Runtime



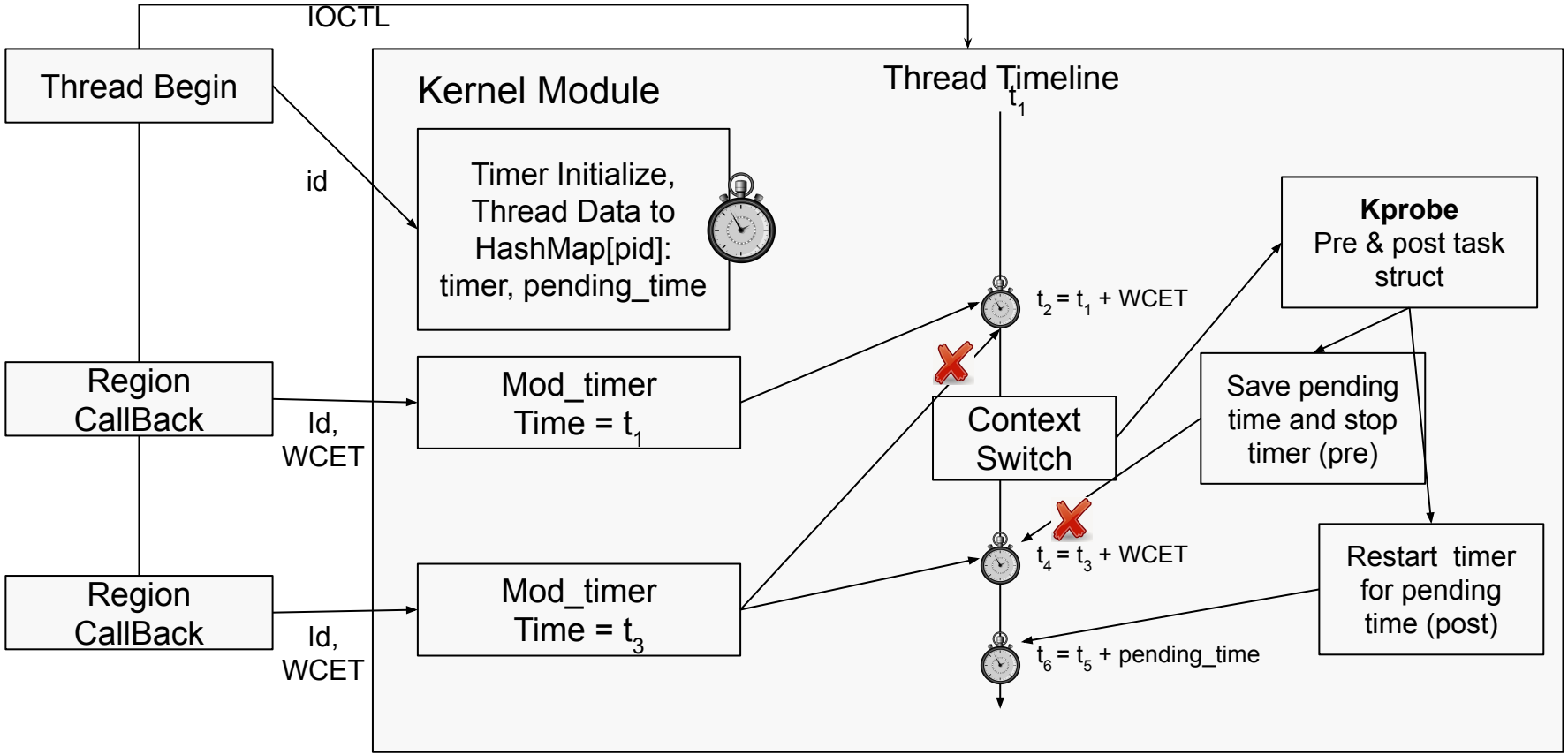
T-Tex Execution Pipeline

- Modified callbacks
- **Data structure information read from file**
- Executing timers at region callback
  - **Record execution time for previous region**
    - Eliminate the timer
    - Start timer for new region
      - **How do we get the timer value?**
  - Callbacks provide region IDs
  - Retrieve time values of region
    → from **data structure (using the ID's)**
- **Executed region information appended**
  - Thread specific data of OMPT
  - Critical for update WCET analysis
  - **How to use this information?**



- Analysis phase
  - WCET of each region calculated, saved in file
  - **Security vulnerability threshold (SVT)**
    → set by the user (E.g., 60us)
- Phase 1 (and then Phase 2,3,...)
  - Loops
    - split_iteration
      - 2 code regions needed to divided 100 us
        - Each code region < 60 us
    - Split instructions if split_iteration = 1
  - Regions
    - 4 code regions needed to split 200 us
    - Total instruction / 4
  - Handling common “func” call
    - **Assign function to region with highest WCET for code division**

Linux Kernel Timer Crediting

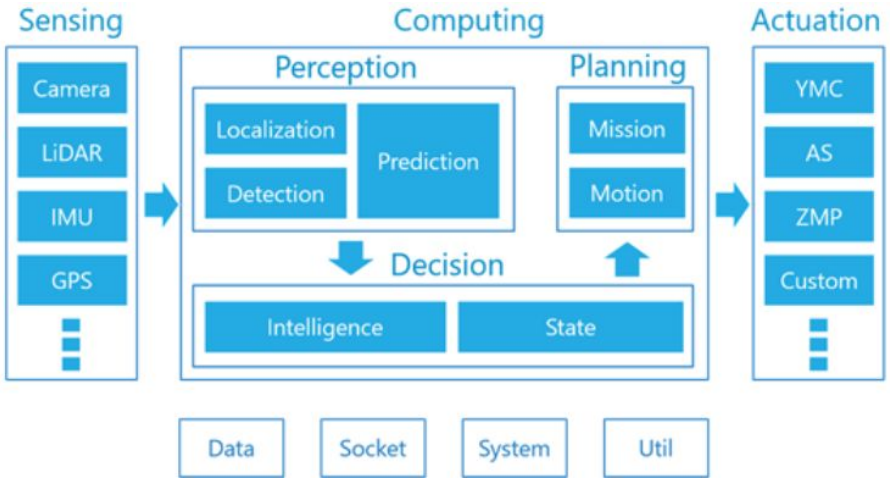


Contents

Evaluate

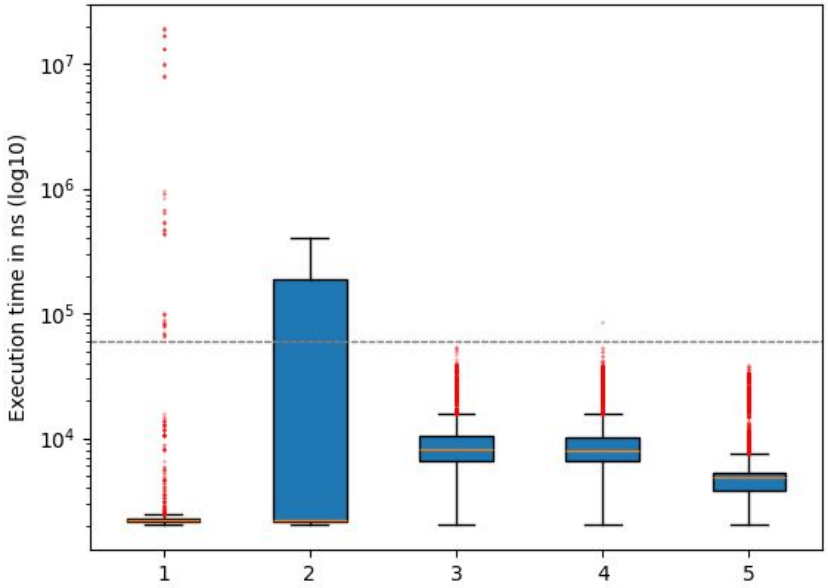
- **Multiphase code division using T-Tex**
  - Execution time below vulnerability threshold
- **Benefit of kernel timer crediting**
- **Accuracy under delay attacks**
  - Chain delay (executing delay in each code region)
- **T-Tex vs. Basic Block level security (~ T-SYS)**
- **Performance overhead**
- System:
  - x86_64 Intel i7-9750H processor at 2.60GHz
  - **OpenMP application executed via 4 threads on 2 cores (oversubscription)**
    - with real-time threading capabilities
      - linux real-time priority band replicating OpenMP-RT

Experiment 1: Daphne Benchmark



- Daphne benchmark
  - Benchmark to evaluate **autoware platform**
    - Autonomous driving framework
  - **Mirrors key modules of perception**
    - Points to Image
    - Euclidean Clustering (Object Detection)
  - **Object detection is a time critical task**
    - Demonstrates real-time task

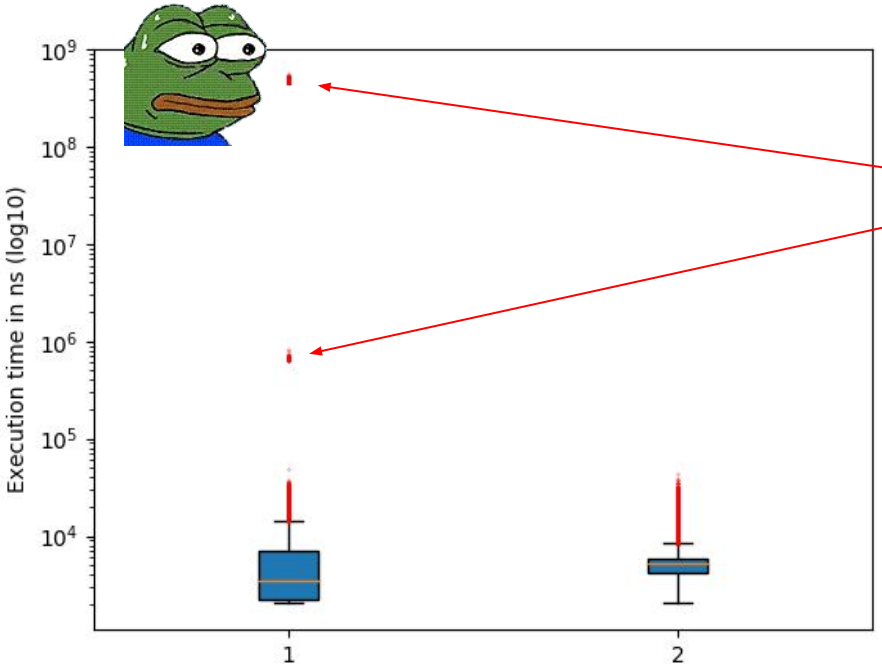
Multiple Phases of T-Tex



- Execution time values of code regions after T-Tex Phase **1. 2. 3. 4. & 5. (Box Plots)**
- Phase 5 achieves requested security threshold

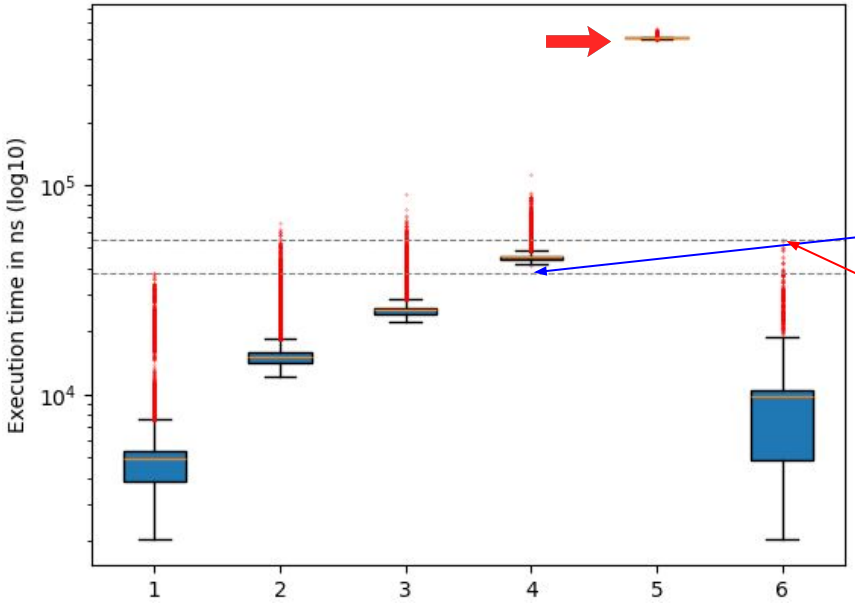
- T-Tex executes 5 phases
  → achieves 60us security threshold
  - 100K code regions
  - Execution of all regions < 60us

Need for Timer Crediting



- 1 - **WCET evaluate with no crediting**
  - T-Tex divides code based on WCET
  - **Impossible to break down regions**
    - **Finer division than instruction level needed**
- 2 - **WCET evaluated with crediting**
  → achieves 60us security threshold
- **Higher vulnerability for 1**
  - Higher WCET

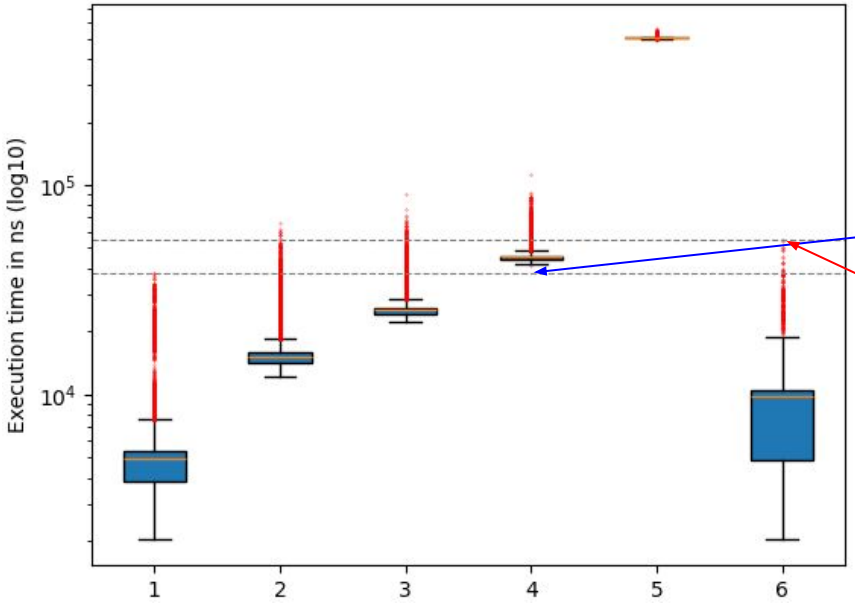
Attack Evaluation



- Execution time values of code regions under delay attack (T-Tex protection)
  - **1. No Delay 2. 10us 2. 20us 3. 40us 4. 500us & 6. ~ T-SYS (No Delay)**

- **Exec. time values under attack delays**
  - Delays in each code region
    - Replicates **chain of delay attacks**
  - **No overlap with 40us delay**
    - See 1st dotted line
- **Basic block level security**
  - **Replicates T-SYS**
    - **Overlaps with 40us delay**
      - See 2nd dotted line
- 100% attack detection for a 500us delay even with phase2 level of security

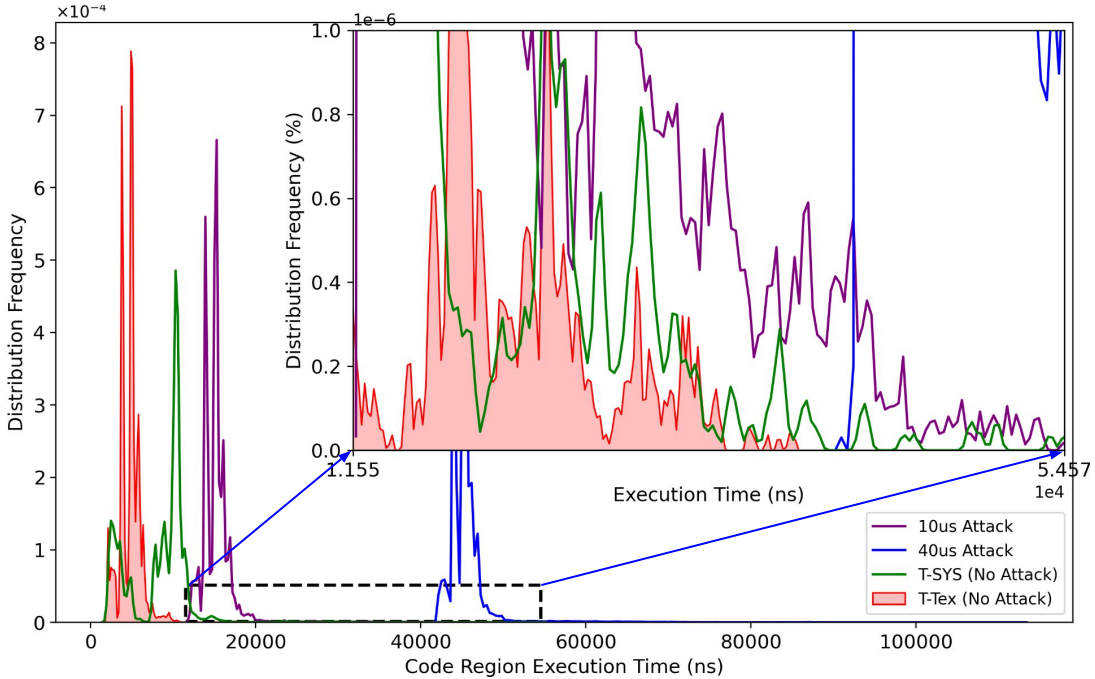
Attack Evaluation



- Execution time values of code regions under delay attack (T-Tex protection)
  - **1. No Delay 2. 10us 2. 20us 3. 40us 4. 500us & 6. ~ T-SYS (No Delay)**

- **Exec. time values under attack delays**
  - Delays in each code region
    - Replicates **chain of delay attacks**
  - **No overlap with 40us delay**
    - See 1st dotted line
- **Basic block level security**
  - **Replicates T-SYS**
    - **Overlaps with 40us delay**
      - See 2nd dotted line
- How many code regions are affected?
  - A single value would result in higher WCET for multiple regions

Normal Distribution of Box Plots - Study Overlapping Regions



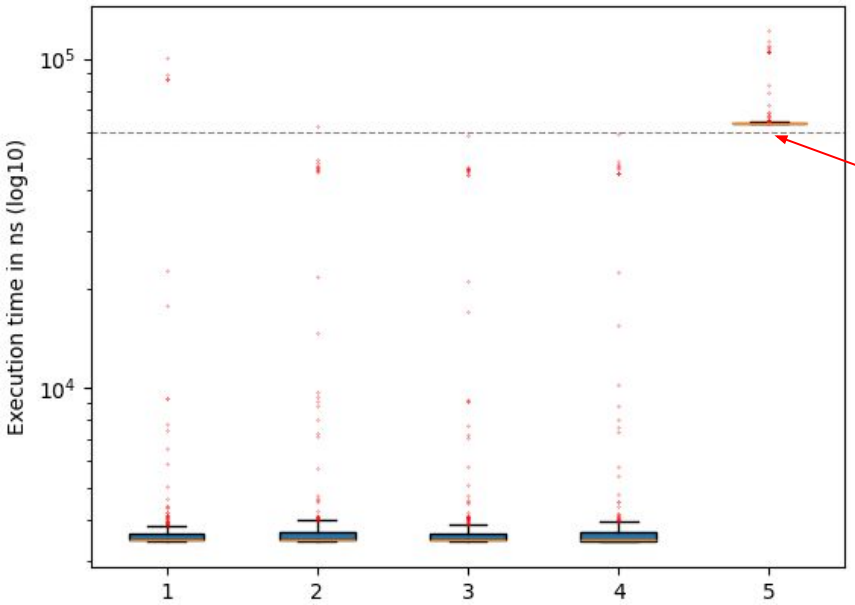
- **0.7% values in red overlap with purple**
  - **Effects 22K regions** (out of 100K regions)
  - **Less than 25% chance** for an attacker to mask a delay
- **0.05% of values in green overlap with blue**
  - No values in red overlap with blue
  - **800 regions**
    - **Higher vulnerability window**

Experiment 2: Parsec

Program	Application Domain	Parallelization		Working Set	Data Usage	
		Model	Granularity		Sharing	Exchange
blackscholes	Financial Analysis	data-parallel	coarse	small	low	low
bodytrack	Computer Vision	data-parallel	medium	medium	high	medium
canneal	Engineering	unstructured	fine	unbounded	high	high
dedup	Enterprise Storage	pipeline	medium	unbounded	high	high
facesim	Animation	data-parallel	coarse	large	low	medium
ferret	Similarity Search	pipeline	medium	unbounded	high	high
fluidanimate	Animation	data-parallel	fine	large	low	medium
freqmine	Data Mining	data-parallel	medium	unbounded	high	medium
raytrace	Rendering	data-parallel	medium	unbounded	high	low
streamcluster	Data Mining	data-parallel	medium	medium	low	medium
swaptions	Financial Analysis	data-parallel	coarse	medium	low	low
vips	Media Processing	data-parallel	coarse	medium	low	medium
x264	Media Processing	pipeline	coarse	medium	high	high

- OpenMP benchmark
  - Freqmine (FMI)
    - frequent itemset mining
  - **Less compute and loop intensive** application
- Evaluate compatibility and performance overhead

Results



- Execution time values of code regions after T-Tex Phase **1. 2. 3. & 4 (Box Plots)**
- **5. 60us delay attack**
- Phase 3 achieves requested security threshold
  - Phase 4 results in no updates

- **3 phases of T-Tex needed for 60us security threshold**
  - Less **code regions (220)**
- **100% detection for 60 us delay attack**
- **11% performance overhead** compared to no protection
  - **71% overhead in case of Daphne benchmarks**
  - **35% for phase 2 level of protection**
    - **100% detection for 500us** delay

Contents

Conclusion

- **Contributed → T-Tex: Timed analysis and attack detection technique**
  - For real-time multiprocessor applications **using OpenMP**
- Leverages the **LLVM compiler and OMPT profiler**
  - implements a multi-timer approach **supporting dynamic timed analysis**
- Successfully **detects 100% of attacks**: **40–60us delays**
- **11-72% overhead** depending on the OpenMP application
- **Novel kernel timer crediting** technique

Q/A?

Thank You!!