Protecting Actuators in Real-Time Cyber-Physical Systems

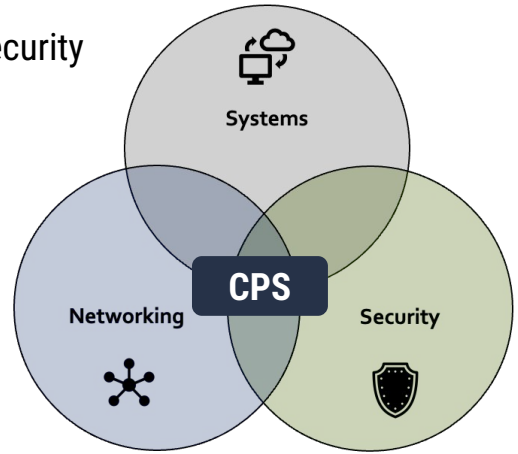
Monowar Hasan
Assistant Professor
School of EECS, WSU

WASHINGTON STATE UNIVERSITY

CySER Workshop 2024

$ whoami

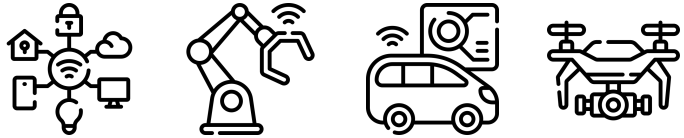
- Assistant Professor
  - EECS@WSU
- Lab:
  - Cyber-Physical Systems Security Research Lab
  - https://cps2rl.github.io
- Research
  - Systems security



- Security for real-time and cyber-physical systems
- Trustworthy ML for embedded/IoT systems
- Resilient real-time networks using SDNs
- Security for precision agriculture

Today's Talk

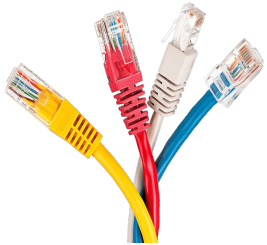
Real-Time Cyber-Physical Systems Security



Cyber-Physical Systems (CPS)

CYBER

Software, Control Algorithms, Code

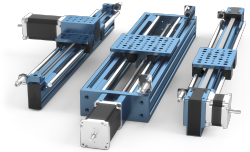


Networking, Communication

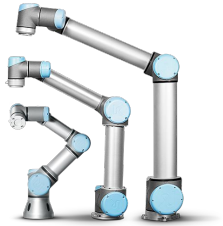
Microcontrollers, ECU, PLC

PHYSICAL

Sensors

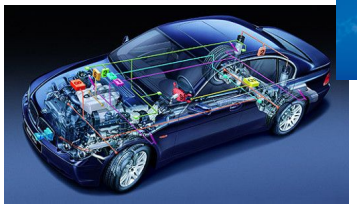


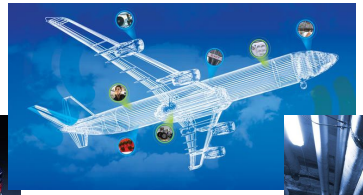
Actuators



Plant

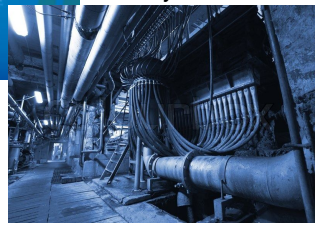
Automobiles





Avionics

Control Systems



CPS Applications

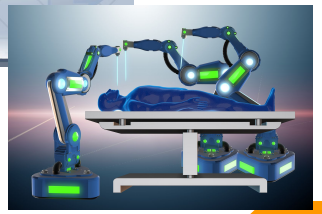
Unmanned Vehicles

Surveillance

Autonomous Driving

Manufacturing

Healthcare



* Image courtesy: Google Image Search

WASHINGTON STATE UNIVERSITY

Traditional RTS

- Custom Hardware
- Proprietary Operating System
- Proprietary Software
- Limited Network Connection

Modern RTS

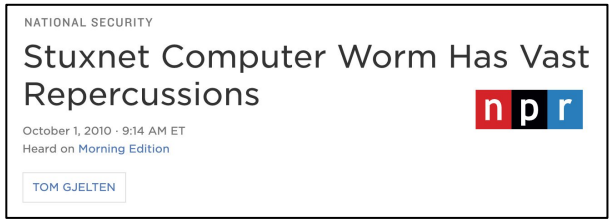
- COTS Hardware
- Open Source Operating System
- Open Source Software
- More Connectivity → Internet!

Larger Attack Surface!

Modern RTS are vulnerable to security threats!

Real-Time CPS Security [?]

→ Increased Security Risks


NATIONAL SECURITY

Stuxnet Computer Worm Has Vast Repercussions

npr

October 1, 2010 · 9:14 AM ET

Heard on Morning Edition

TOM GJELTEN


WIRED Hacker Says He Can Hijack a $35K Police Drone a Mile Away

ANDY GREENBERG SECURITY 03.02.16 09:00 AM

Hacker Says He Can Hijack a $35K Police Drone a Mile Away


THE DRIVE | THE WAR ZONE | MOTORCYCLES | REVIEWS

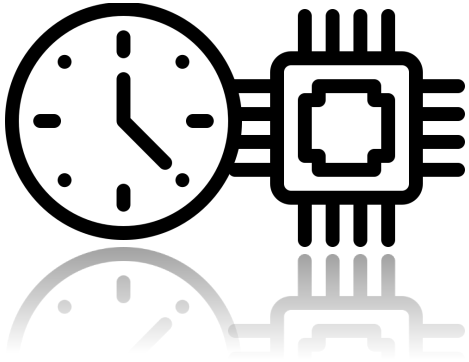
Hacker Claims Ability to Remotely Shut Off Car Engines While Vehicles Are in Motion

It's getting easier and easier to hack a car. Are we on the verge of a dangerous nightmare?

BY JONATHON KLEIN APRIL 30, 2019

Real-Time CPS

Distinguishing Properties

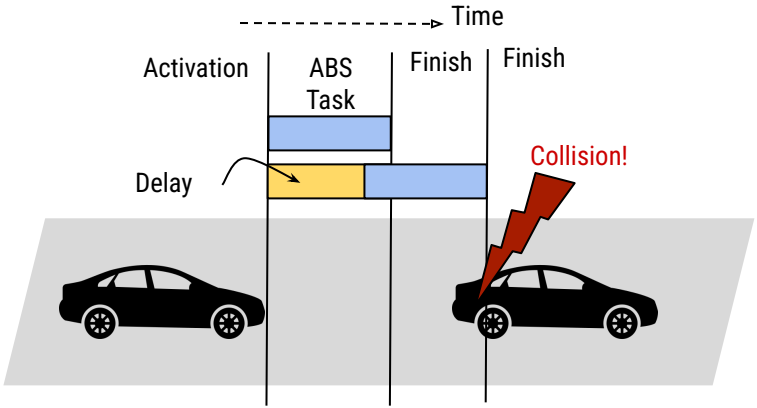


Critical Parameter: Time

Real-Time Systems (RTS) → requires both logical and temporal correctness

- Real-time ≠ fast
- Timing constraints → measured by deadlines

- Example:
  Anti-lock Braking System (ABS)



Anti-lock Braking System (ABS) in cars

- must function correctly in milliseconds
- even 1 second delay might be too late
- car traveling at 60 mph → 88 ft. in 1 s!

Real-Time Requirement

Normal vs Late Airbag Deployment

Normal Deployment

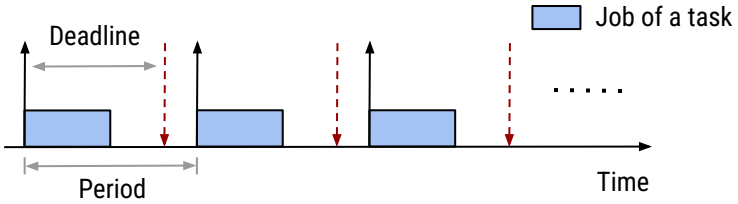
Late Deployment

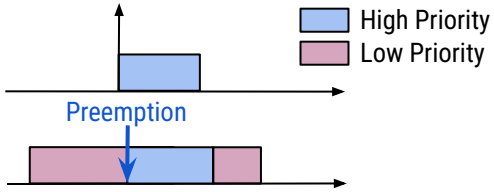
Perturbation of timing constraints → serious consequences

Source: https://www.youtube.com/watch?v=YAwrq9-1oQQ

Real-Time CPS: Common Properties

- Limited Resources
  - ↪ computational power, energy
- Periodic Tasks
- Priority-driven scheduling

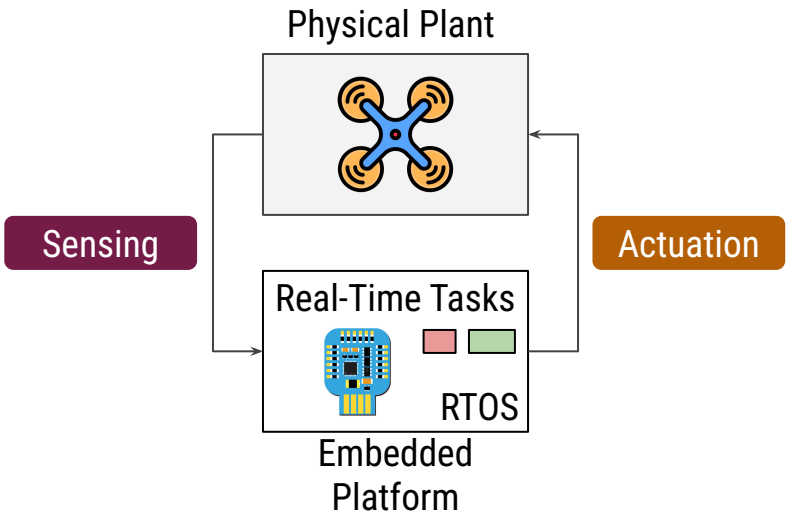


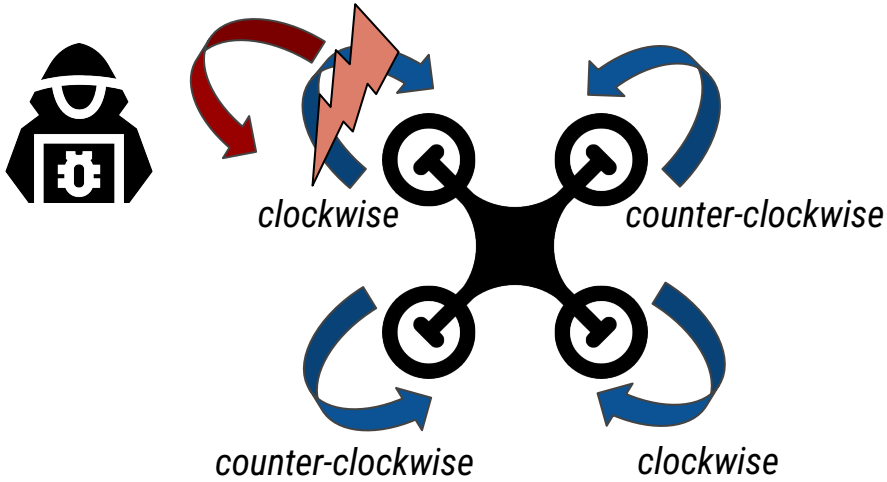


Real-Time CPS: Common Properties

- Real-time CPS → based on **sensing** and **actuation**

false/spoofed actuation command → disrupt normal operation





Today's Talk

> ***How to prevent falsification of actuation commands?***

Design Goals

- Examine actuation commands
  - **before** they are being issued to actuators
- Guarantee timing requirements
- Tamper-proof
- Better compatibility
  - design with off-the-shelf components

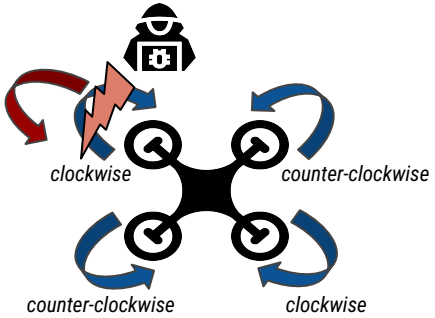
Trusted Execution Environments (TEE)

arm TRUSTZONE

Model and Assumptions
System Model

- A set of fixed-priority, periodic real-time tasks
- Multicore platform, partitioned scheduling
- Each task *i* generates N_i actuation requests

Model and Assumptions
Adversary Model



- Adversarial actions result in modification of actuation commands

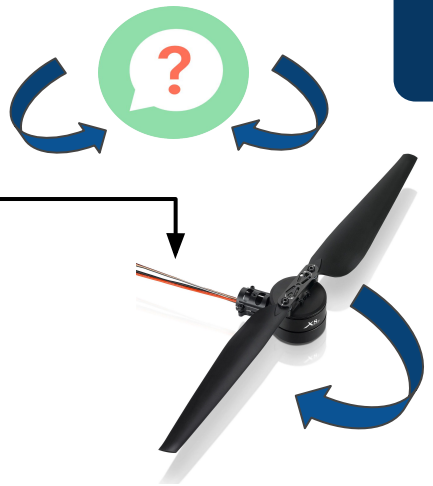
- No assumptions on how adversary compromises tasks
  - Known vulnerabilities
  - Remote trozans
  - Social engineering
  - ….

- No physical presence
  - Can not physically control/turn off/damage actuators

Vanilla (Non-Secure) Execution

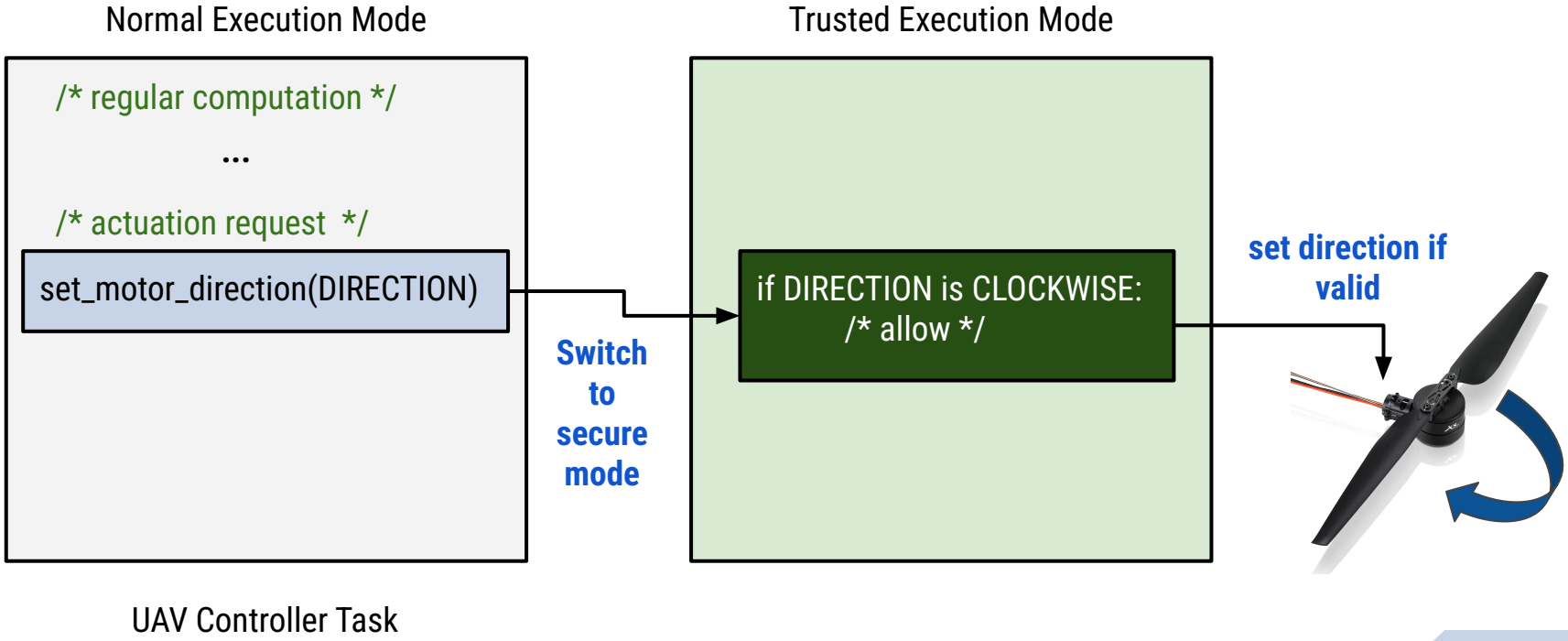
```
/* regular computation */
.
.
.
/* actuation request  */
set_motor_direction(DIRECTION)
...
...
```

UAV Controller Task



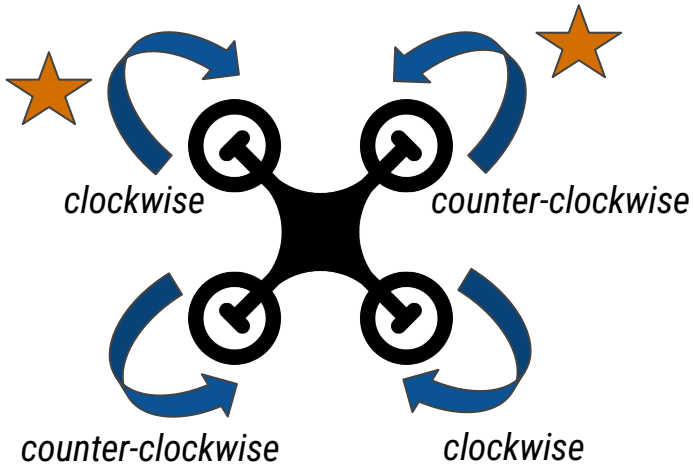
How to prevent the sending of malicious commands to actuators?

Proposed Scheme: Overview



Checking of Actuation Commands

- Designer-provided, design-time **“rules”** → required for correct system operations
  - maps *state* → *action*



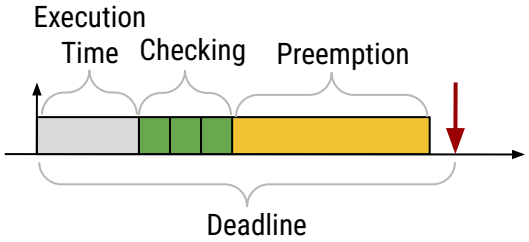

```
If MOTOR is FRONT_LEFT:
     DIRECTION = CLOCKWISE

If MOTOR is FRONT_RIGHT:
     DIRECTION = ANTICLOCKWISE

....
```

But...

we can’t always check what we wanted!

Real-time constraints!

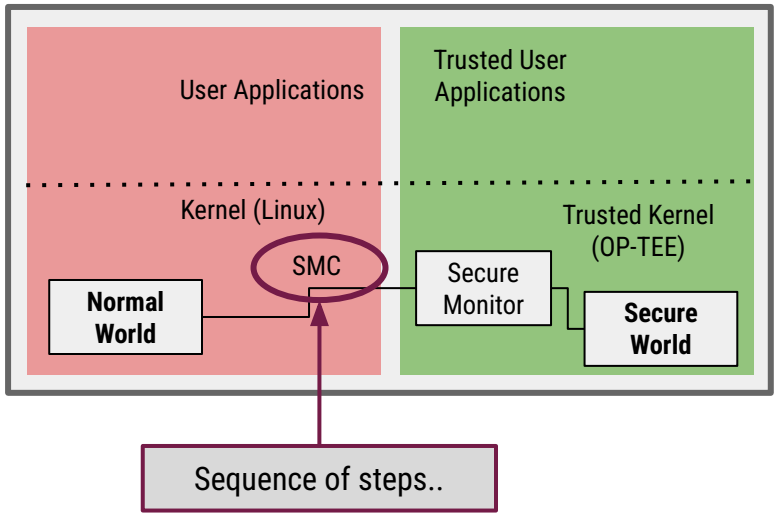
- Timing constraints?



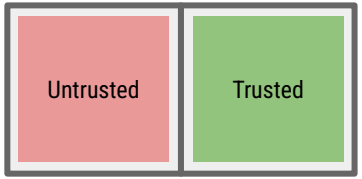
Execution Time + Checking Overhead + Preemption ≤ Deadline

NOT insignificant for TEEs!

Background - ARM TrustZone



arm TRUSTZONE → isolates trusted software and data



Normal to secure mode context switch is costly!

OP-TEE

Raspberry Pi 3

Mode switching steps:

			Overhead (microsec)
1.	Initialize	`TEEC_InitializeContext()`	64
2.	Open session	`TEEC_OpenSession()`	49233
3.	Transfer	`TEEC_InvokeCommand()`	146
4.	Close session	`TEEC_CloseSession()`	15682
5.	Clean up	`TEEC_FinalizeContext()`	29

Total Overhead: **66** ms

⇧ for multiple commands

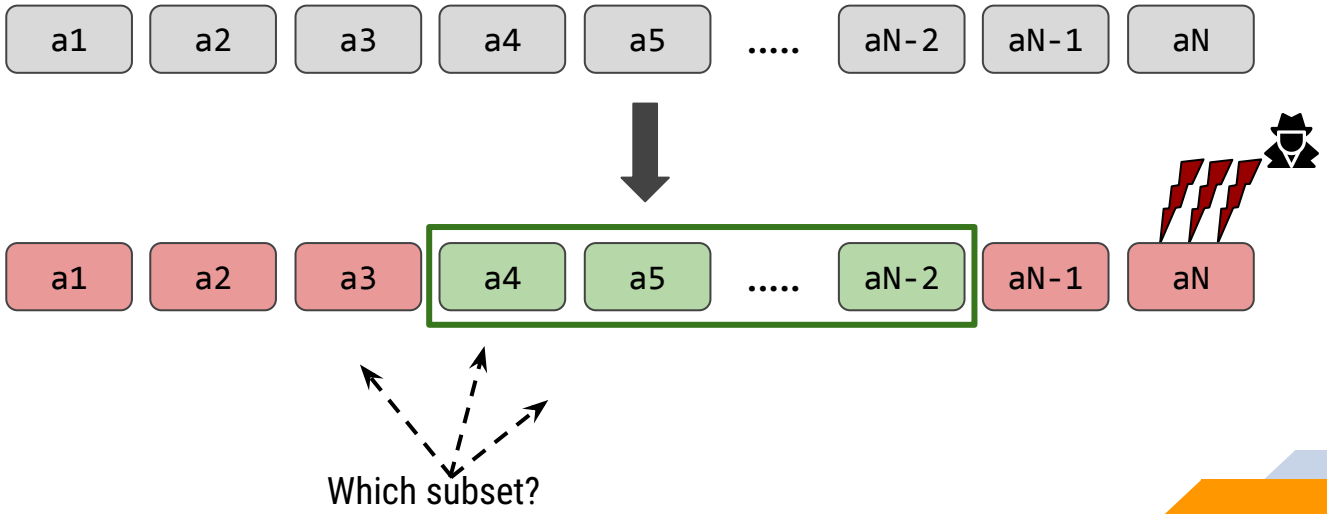
Require "selective" checking!

Ref: rover/drone operates at 5 Hz (200 ms)

How to reduce checking overheads?

Can we do better?

- Check a subset of actuation commands



Our Approach
SCATE

Selective Checking And Trusted Execution

M. Hasan and S. Mohan, "You Can't Always Check What You Wanted: Selective Checking and Trusted Execution to Prevent False Actuations in Real-Time Internet-of-Things," *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, Nashville, TN, USA, 2023, pp. 42-53, doi: 10.1109/ISORC58943.2023.00017

SCATE: Key Idea

- For each job of a task:
  - ***non-deterministically*** select a **subset of commands** for checking

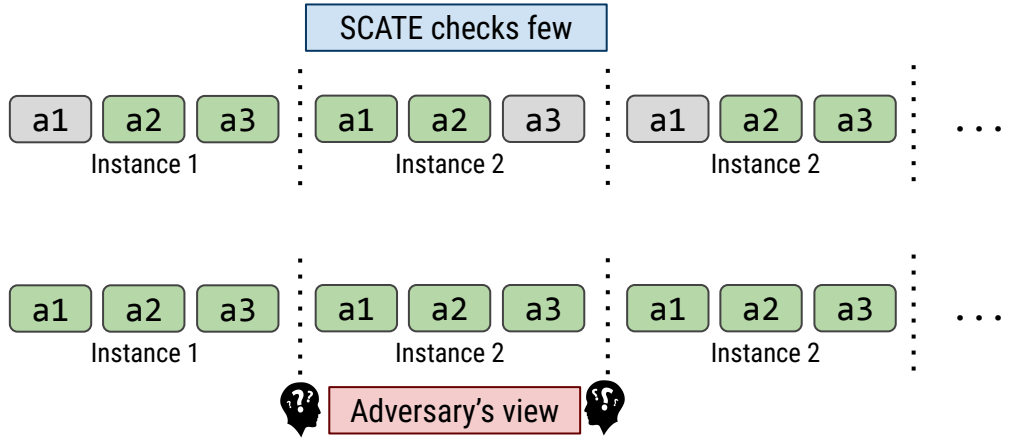
a1 a2 a3

SCATE checks few commands

But..

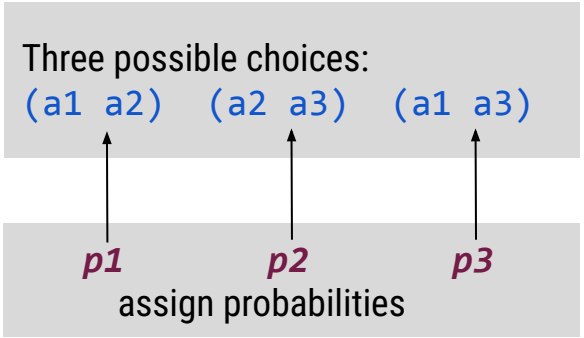
From adversary's view:

SCATE checks it all!



SCATE Example

- Let:
  - A task generates **3** actuation commands `a1 a2 a3`
  - We can only check **2** commands
- Goal:
  - Check ***all*** commands → adversary's view



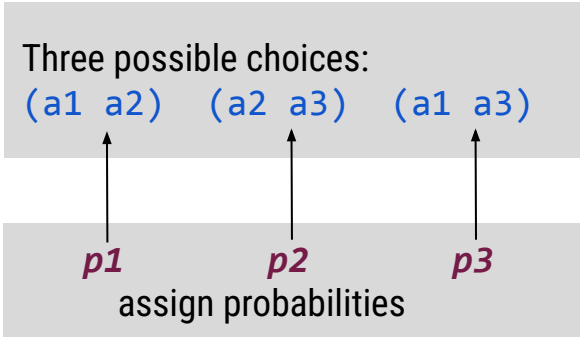
- Runtime random selection:
  - Let ***p1***= 0.5, ***p2***= 0.4, ***p3***= 0.1

- *Fitness Proportionate Selection*
  - Generate random number *R* in [0, 1]
    - If *R* in [0, 0.5) → check `(a1 a2)`
    - If *R* in [0.5, 0.9) → check `(a2 a3)`
    - If *R* in [0.9, 1.0] → check `(a1 a3)`

SCATE Example

- Let:
  - A task generates **3** actuation commands `a1 a2 a3`
  - We can only check **2** commands
- Goal:
  - Check ***all*** commands → adversary's view

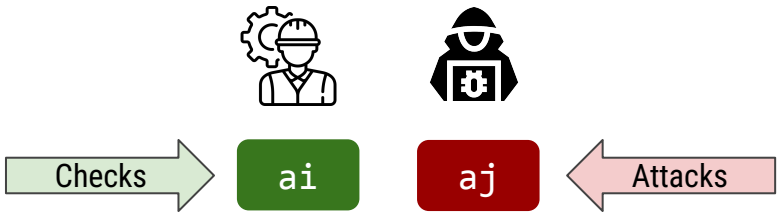
How to derive these probabilities?



- **Game theoretical analysis**
  - Two-player game
  - Linear optimization problem

SCATE
Game Theoretical Analysis

- Two-player game
  - **Designer** and **Attacker**



- Parameters:
  - System Reward
  - System Cost

- **Reward**$_{ij}$
- **Cost**$_{ij}$

- ⇧ reward & ⇩ cost ⟶ **GOOD** for designer, **BAD** for attacker
- ⇩ reward & ⇧ cost ⟶ **BAD** for designer, **GOOD** for attacker

Formulate linear optimization problem

- maximize reward, minimize cost
  ⟶ probabilities of checking commands

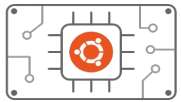
Evaluation

SCATE Implementation:
https://github.com/mnwrhsn/scate_implementation

- Implementation:
  - Raspberry Pi 3
  - Linux 4.16.56
  - OP-TEE 3.4

Raspberry Pi 3

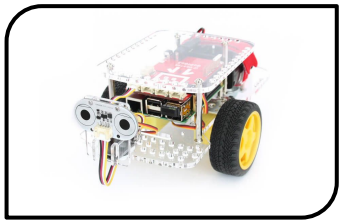
OP-TEE Secure OS



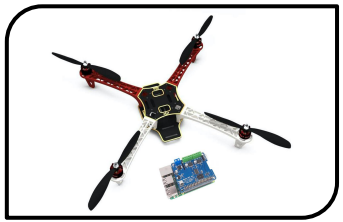
Embedded Ubuntu Linux

- RT-IoT Platforms:
  - COTS hardware

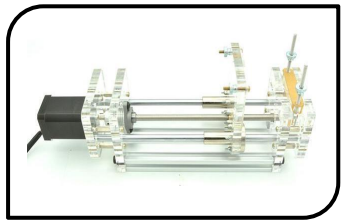
Ground Rover



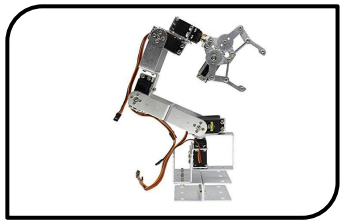
Flight Controller



Syringe Pump



Robot Arm



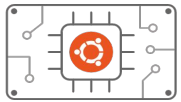
Evaluation

SCATE Implementation:
https://github.com/mnwrhsn/scate_implementation

- Implementation:
  - Raspberry Pi 3
  - Linux 4.16.56
  - OP-TEE 3.4

Raspberry Pi 3

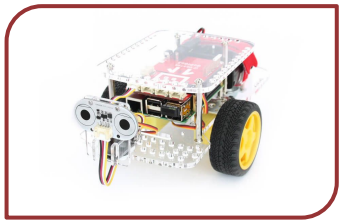
OP-TEE Secure OS



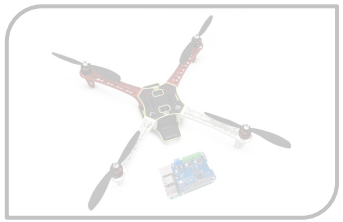
Embedded Ubuntu Linux

- RT-IoT Platforms:
  - COTS hardware

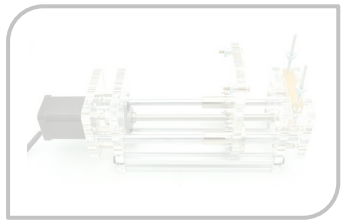
Ground Rover



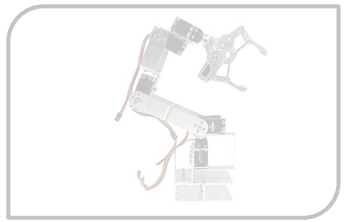
Flight Controller



Syringe Pump



Robot Arm



Evaluation
Actuation Commands, Checks Inside Enclave, and Attacks

Ground Rover

Actuation Commands

- Set the speed of the wheels
- Set direction of the wheels (forward, backward, left, right)

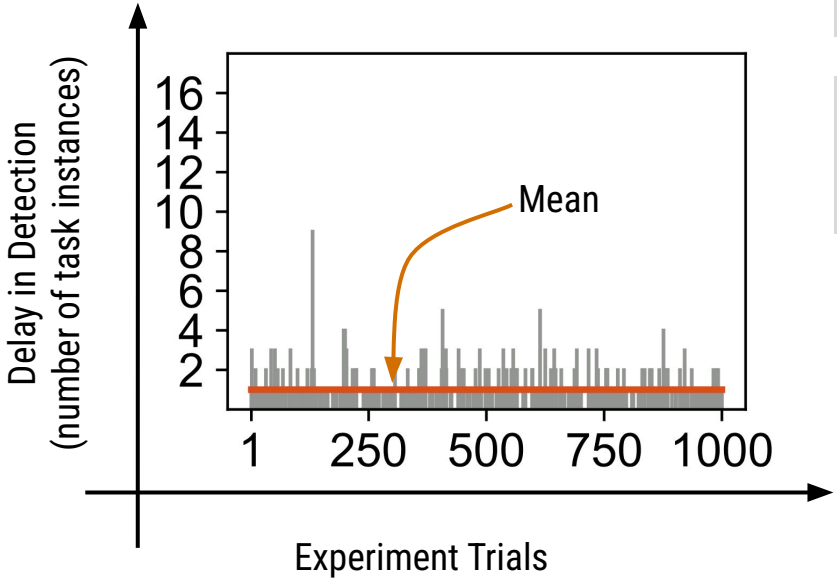
Security Checks

- Speed of the motors should be within predefined range (70-100)

Attacks

- DoS attack → sets arbitrary high speed to the wheel motors
- Destabilize rover

Results
Impact on Detection Time

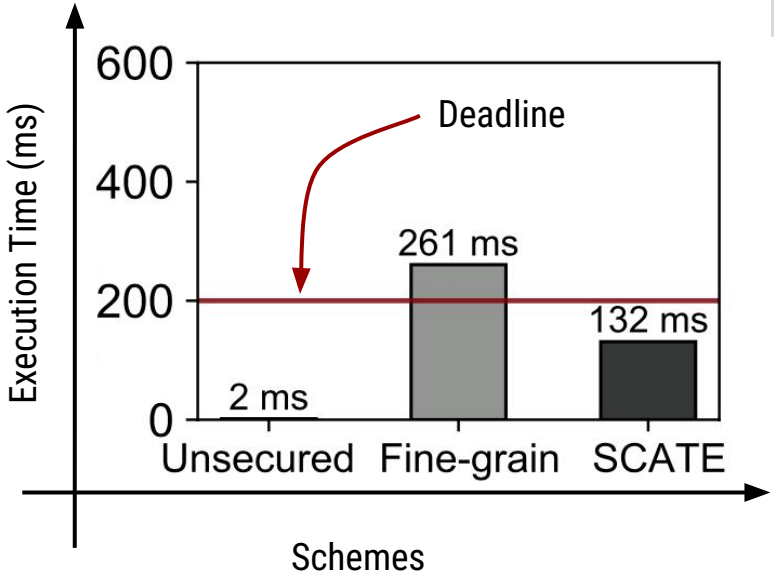


Benchmark scheme:
"Fine-grained" checking → checks **all** four commands

Metric: ***time-to-detect attacks***
→ *how many additional task instances SCATE requires when compared to Fine-grained checking*

SCATE does **not** delay significantly
* Mean delay: **1** additional instances
* 99-th percentile delay: **3** additional instances

Results
Impact on Execution Time

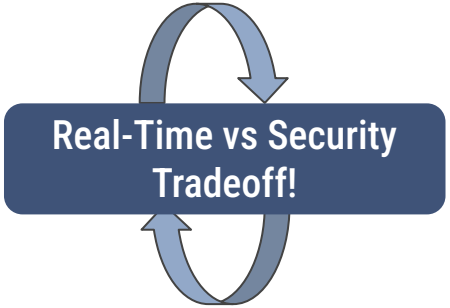


Benchmark schemes:

- ***"Fine-grained"*** → checks **all** four commands
- ***"Unsecured"*** → vanilla execution (**no** checks)

SCATE finishes execution **before** deadline

Fine-grained checking misses deadline!



Experiments | Key Findings

- SCATE → on average requires **1-3 additional instances** to detect attacks
- Fine-grained checking **does not** comply with timing requirements
- SCATE manages to complete **before** deadlines

Remarks

- SCATE:
  - Prevents falsification of actuation commands
    - Trusted Execution Environments → ARM TrustZone
  - Complies with real-time requirements → “selective checking”

✓ Implemented and evaluated on **four** COTS platforms

Thanks!

Questions?

https://monowarhasan.info/

https://cps2rl.github.io

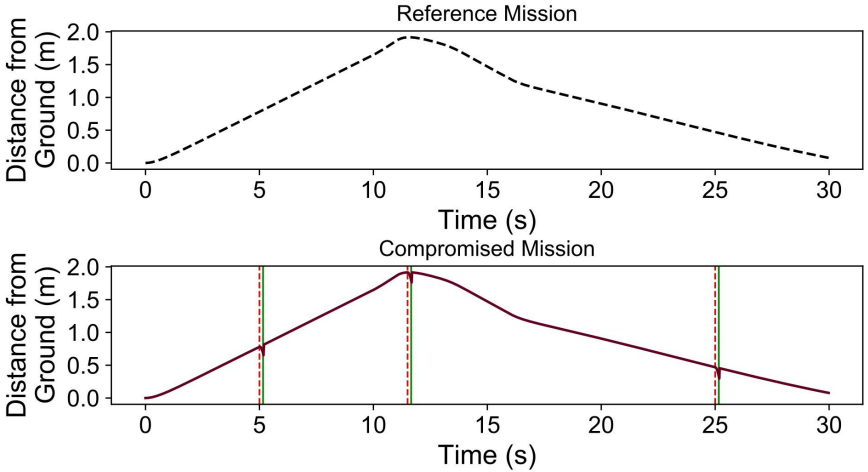
Supplementary Slides

Discussion #1

- How to check *all* commands but minimize TEE overheads?
  - Group multiple commands and check together
    - SCATE *(partially)* supports this
  - Additional evaluation:
    - When we can or can't use this feature

Discussion #2

- SCATE causes some delay in detection →
  - System may not be in “unsafe” state *immediately* due to inertia
  - Demonstrate by experiments



Discussion #3

- 66 ms TEE overhead → platform specific
  - Raspberry Pi 3 → older, slower processor (700 MHz)
  - OP-TEE follows GlobalPlatform API standards
  - Similar findings in prior work*

* Amacher et al, On the Performance of ARM TrustZone, DAIS 2019

On The Performance of ARM TrustZone*
(Practical Experience Report)

Julien Amacher and Valerio Schiavoni

Université de Neuchâtel, Switzerland, first.last@unine.ch

~900 ms!

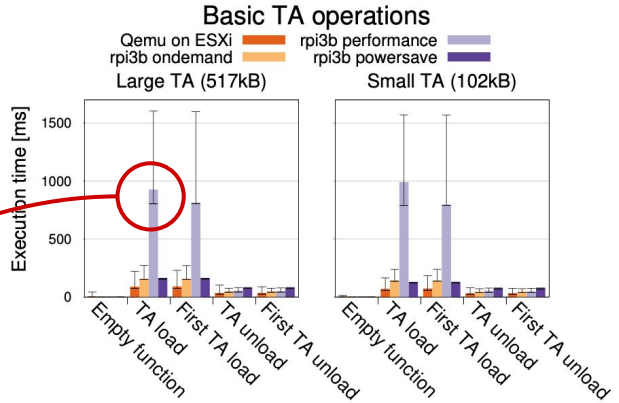
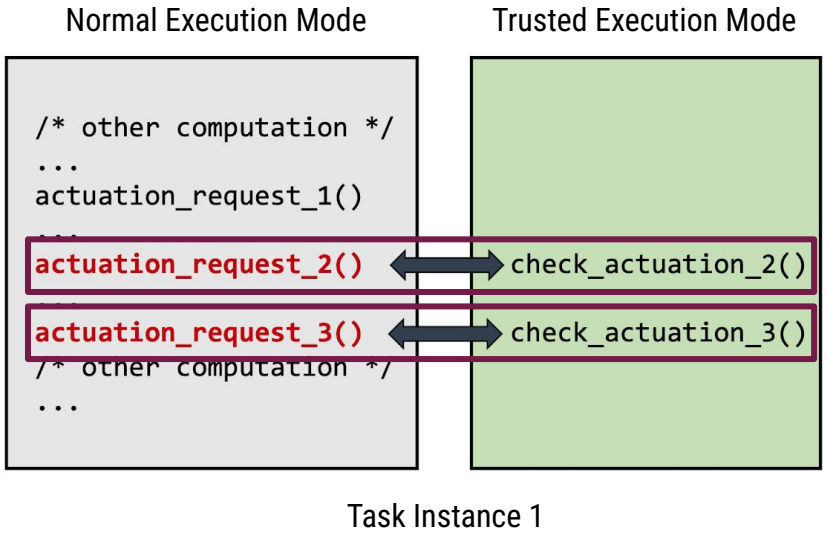


Fig. 7: Basic TA operations: loading, unloading and successive calls to load/unload the same TA.

SCATE
Illustration

- Let:
  - A task generates **3** actuation commands
  - We can only check **2** commands



Task Instance 1

Evaluation Platforms
Summary

Flight Controller

Actuation Commands

- Set PWM frequency
- Set PWM pulse durations wheels (for four motors)

Security Checks

- Check PID control coefficients

Attacks

- Set incorrect PWM pulse values
- Destabilize quadcopter

Robot Arm

Actuation Commands

- Set rotation angle of the arm
- Four PWM pulses

Security Checks

- Check robot arm can move up to certain angle

Attacks

- Synchronization attack → sets incorrect angle value
- Destabilize robot operation

Syringe Pump

Actuation Commands

- Set motor rotation frequency
- Push/pull motors (3 each)

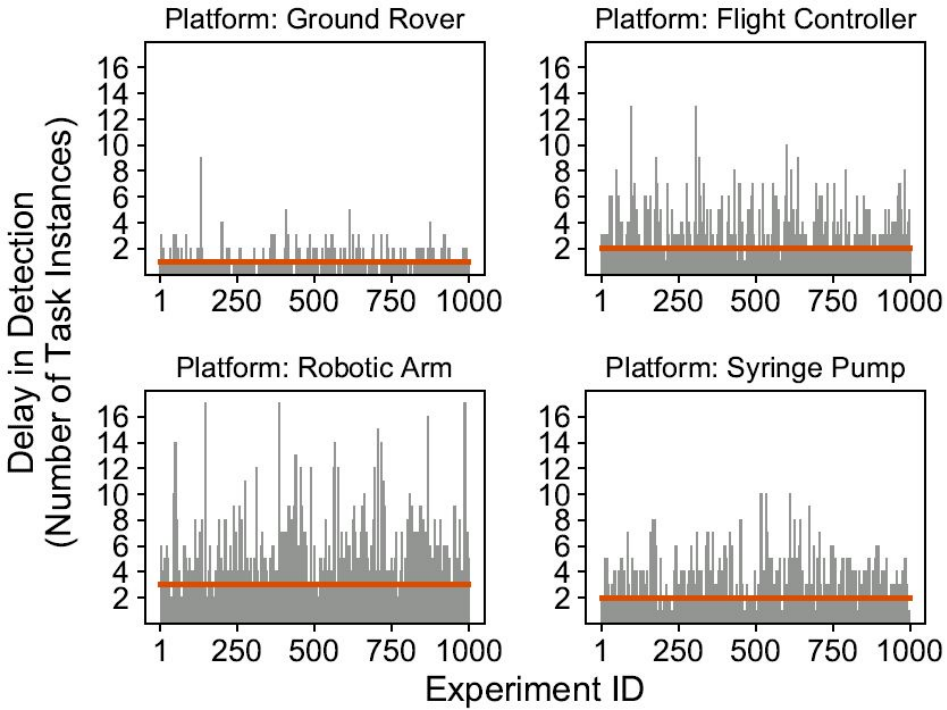
Security Checks

- Check number of push/pull operations

Attacks

- Injects more fluid than desired
- Health/safety concern

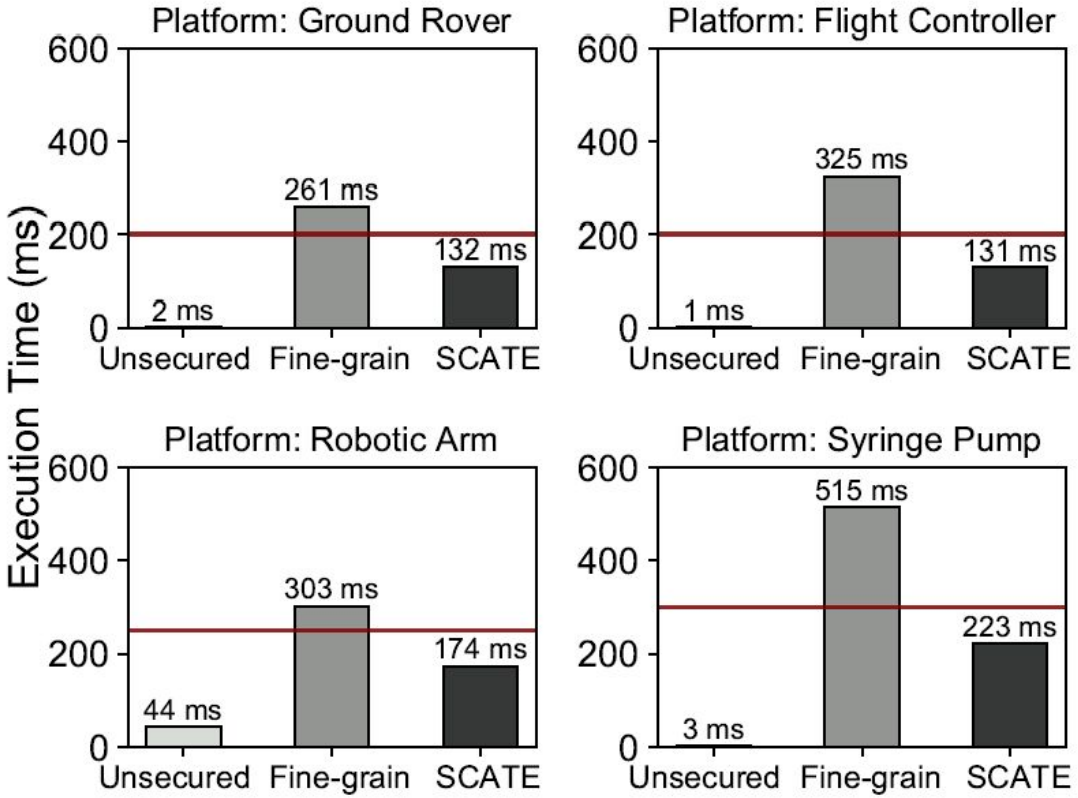
Results
Impact on Detection Time



Benchmark scheme:
"Fine-grained" checking → checks **all** commands

Metric: ***time-to-detect attacks***
→ *how many additional task instances SCATE requires when compared to Fine-grained checking*

Results
Impact on Execution Time



Results (contd.)

Impact on Detection Time

Platform	Delay	
	Mean	99th percentile
Flight Controller	2	8
Robot Arm	3	12
Syringe Pump	2	8

Mean delay is no more than **3** instances

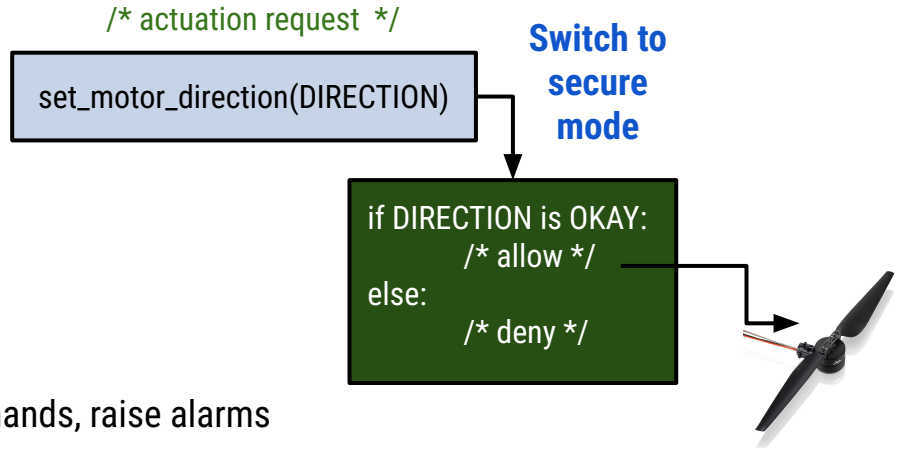
Impact on Detection Time

Platform	Deadline (ms)	Execution Time (ms)	
		SCATE	Fine-Grain
Flight Controller	**200**	**131**	**325**
Robot Arm	**250**	**174**	**303**
Syringe Pump	**300**	**223**	**515**

SCATE complies with timing requirements
Fine-grained checking misses deadlines!

Limitations and Discussion

- Some attacks may not be detectable
  - Zero-day attacks?



- SCATE blocks malicious commands
  - Other possibilities → send buffered commands, raise alarms