Coping with Security / Safety Tensions in Low-End Embedded Devices

Gene Tsudik
CS Dept., UC Irvine

Joint work with: X. Carpent[1], K. Eldefrawy[2], N. Rattanavipanon[1], A. Sadeghi[3]
1 – UC Irvine, 2 – SRI International, 3 – TU Darmstadt

1

Roadmap

- Overview of Remote Attestation (RA)
- Problem Statement
  - Conflict between security of RA and real-time operation
- Tentative mitigation measures
  - Periodic self-measurements
  - Interruptible RA with shuffled measurements
  - Interruptible RA with memory locking
- Conclusions & future work

2

Internet-of-Things (IoT) Gadgets

3

IoT-specific Attacks On:

- Sensing: Privacy
- Actuation: Security & Safety
- Either: DDoS Sourcing (aka Zombification)

4

Constraints for Simple IoT Devices: large scale + low price

CPU Power	Battery

Hard to prevent malware from entry

Next best thing is to detect malware!

Memory Size	Hardware Cost

Plus, limited physical "real estate" and typically low security budget

5

Remote Attestation (RA)

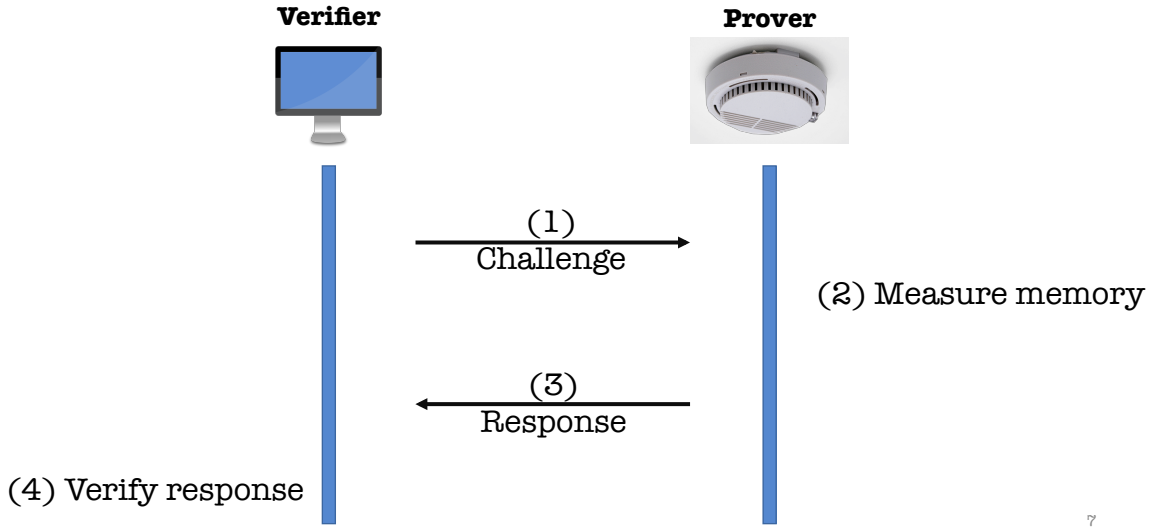
- A means of detecting malware presence on an untrusted remote device
- A security service
- A protocol
- An architecture

All of the above…

- RA typically realized as an interaction between:
  - **Verifier**: trusted entity
  - **Prover**: untrusted (potentially infected) remote device
- Goal: learn current internal (sw/fw) state of prover

6

Typical (on-demand) RA



7

RA Adversarial Model (DAC'16)

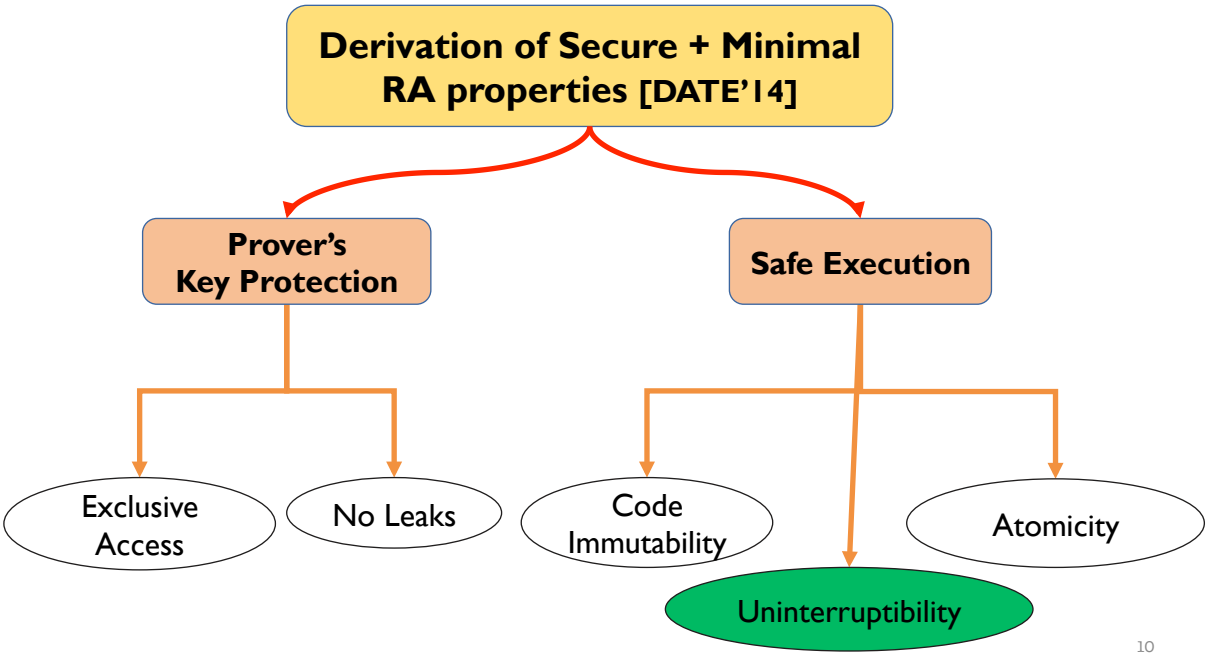
- Remote adversary
  - Exploits vulnerabilities to inject malware
- Local adversary
  - Controls communication channel(s)
- Physical adversary
  - Non-Intrusive: side-channel attacks
  - Stealthy Intrusive: "read-only"
  - Intrusive: alters hardware

8

Landscape of RA Techniques

- Purely Hardware-based
  - Dedicated hardware support, e.g., TPM, TrustZone, SGX
  - Overkill for simple or low-end IoT devices (cost and features)
- Purely Software-based
  - Relies on precise timing measurements, e.g., SCUBA, VIPER, PIONEER
  - Unrealistic assumptions for IoT except peripheral/legacy devices
- Hybrid
  - SW/HW co-design, e.g., SMART, TrustLite
  - Minimal hardware impact
  - **Seems like good fit for low-end devices**

9



10

Hybrid RA Techniques

- SMART [NDSS'12]+[DATE'14]
  - First hybrid design of RA for low-end microcontrollers
- TrustLite [EuroSys'14]
  - Supports secure interrupts
- TyTan [DAC'15]
  - TrustLite with real-time functionality
  - Process being measured cannot interrupt
- HYDRA [WiSec'17]
  - SMART implementation for medium-end devices (secure boot needed)
  - Formally verified *seL4* microkernel guarantees security properties
    - Especially, isolation
- VRASED ['18]
  - First formally verified RA design
  - Based on a version of SMART

11

RA ***vs*** Safety-Critical Operation

on SMART-based MSP430 @ 8MHz
RA takes:

≈4.5 seconds

to measure 48KB of flash

on HYDRA-based ODROID @ 2GHz
RA takes:

≈7 seconds

to measure 1GB of RAM

RA can interfere with safety-critical operation!

So… can we make RA interruptible?

12

Whither Interruptible RA Execution?

- Susceptibility to transient malware
  - Interrupts and erases itself during attestation
  - Avoids detection, leaves no trace
- Self-relocating malware
  - Interrupts and moves itself around during attestation
  - Avoids detection, remains on Prover
- Temporal inconsistency?
  - Memory can change during attestation
  - Computed measurement (e.g., MAC, HASH-to-be-signed) might reflect memory that never existed
  - Could be caused by malware or even benign software
  - An important issue **beyond** the RA context

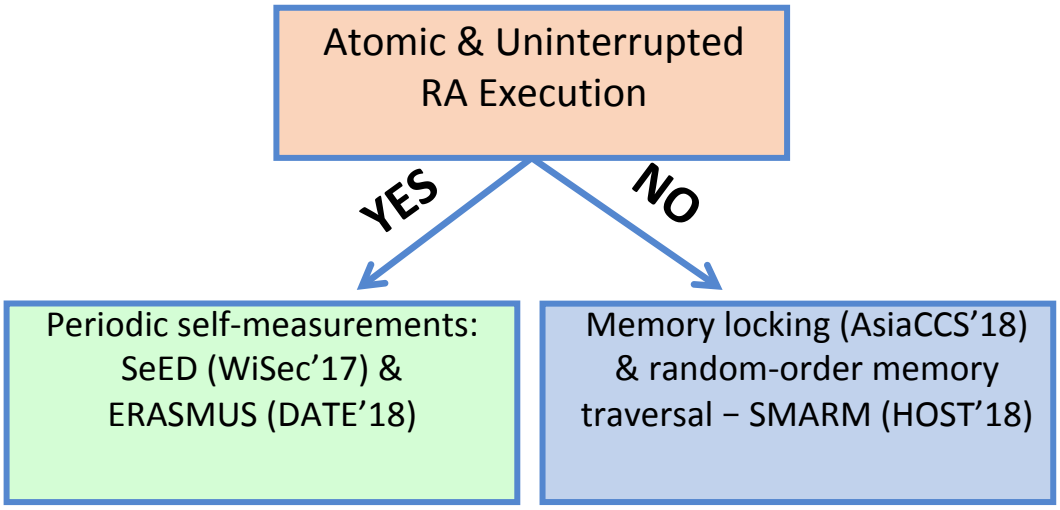
13

TyTan: First Attempt

- Dynamically configurable execution-aware memory protection unit (EA-MPU) enforces:
  - Access rules to prover's attestation key
  - Immutability
- Real-time OS
  - Process can issue interrupt if it's not being attested
  - Provides isolation between processes
- Problems:
  - What if Verifier wants to attest a safety-critical process?
  - OS might be buggy → OS compromise can violate isolation → malware can move around during attestation

14

Solution Landscape



15

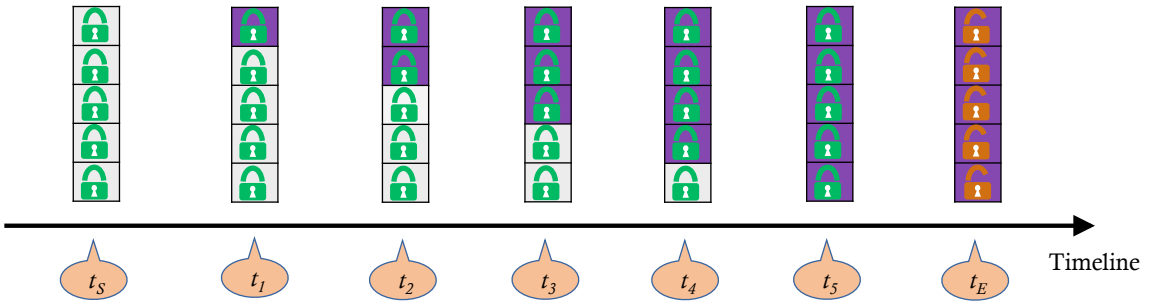
Memory Locking [AsiaCCS'18]

- Can we allow RA measurement process to be interruptible?
- While enforcing ***temporal consistency*** of measured memory
  - E.g., by locking memory segments = temporarily make them read-only
- Measurement (on prover) starts at time: t_S and ends at time: t_E

16

Memory Locking: All-Lock

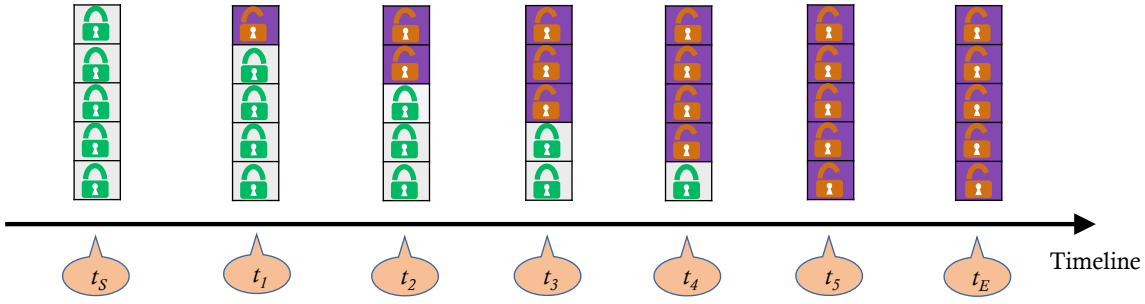
- All memory is locked throughout attestation
- Consistency from t_S to t_E



17

Memory Locking: Dec-Lock

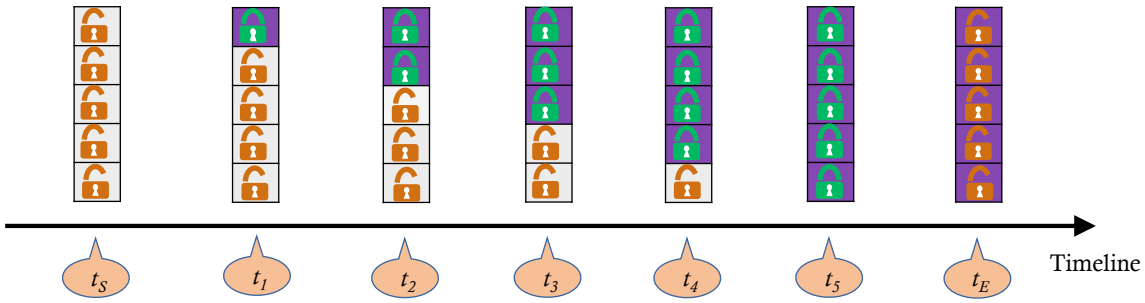
- All memory locked at the start
- A memory block is *unlocked* after it is measured
- Consistency at t_S



18

Memory Locking: Inc-Lock

- All memory unlocked at the start
- A memory block is *locked* after it is measured
- Consistency at t_E



19

Memory Locking: Feature summary

- Attestation process can be interrupted at any time (@block granularity)
- Memory gradually locked or unlocked during attestation
  - Guarantees temporal consistency
  - Detects transient and self-relocating malware
- Requires hardware support to lock/unlock memory blocks
  - MMU (as in HYDRA)
  - Runtime-configurable MPU (as in TyTan)
- Other approaches possible, e.g., inconsistency detection via dirty bits

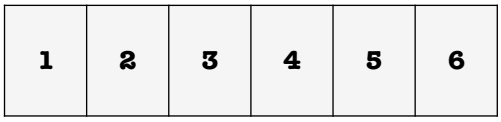
20

Shuffled Measurements [HOST'18]

- Allows attestation process to be interrupted
- Memory traversed in *unpredictable* AND *secret* fashion

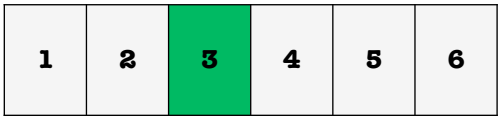
21

Shuffled Measurements



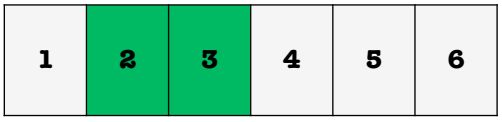
Permutation = {3, 2, 5, 6, 1, 4}

Shuffled Measurements



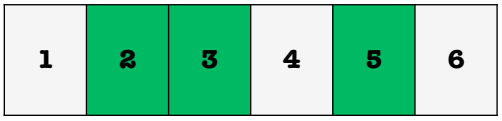
Permutation = {3, 2, 5, 6, 1, 4}

Shuffled Measurements



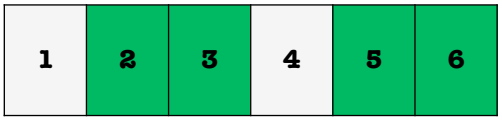
Permutation = {3, 2, 5, 6, 1, 4}

Shuffled Measurements



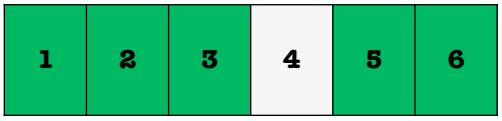
Permutation = {3, 2, **5**, 6, 1, 4}

Shuffled Measurements



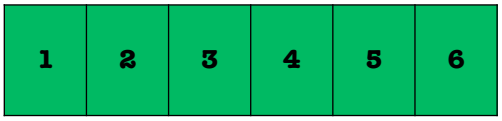
Permutation = {3, 2, 5, **6**, 1, 4}

Shuffled Measurements



Permutation = {3, 2, 5, 6, **1**, 4}

Shuffled Measurements



Permutation = {3, 2, 5, 6, 1, **4**}

Shuffled Measurements [HOST'18]

- Allow attestation to be interrupted @block granularity
- Memory traversed in *unpredictable* AND *secret* fashion
- One-time traversal permutation generated by PRF on input of:
  Key, counter and/or clock, challenge
- Permutation protected by:
  - Storing it in separate (secure) working memory → new feature

  OR
  - Storing encrypt-then-mac of each permutation element in regular memory
- For large number of blocks, 37% malware evasion rate (see paper)
  - Multiple independent measurements to reliably detect malware

29

Shuffled Measurements: Summary

- Attestation process can be interrupted
- Measures memory in random & secret order
  - Probabilistic detection of self-relocating malware
- Multiple measurements to ensure high detection rate
  - Extra overhead in attestation run-time
- Needs no additional hardware features (over basic RA)
  - Encrypt-then-mac each element in permutation

30

Self-Measurement

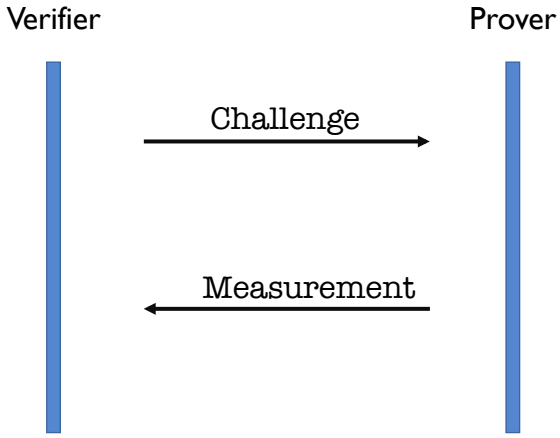
- **Prover** periodically measures own memory:
  - Current RA techniques work "on-demand", i.e., based on **Verifier**'s explicit request

BUT:

- Prover knows best ➔ can schedule measurements to avoid interference with safety-critical tasks
- Attestation process remains uninterruptible
- Two recent techniques:
  - SeED [WiSec'17]
  - ERASMUS [HOST'18]

31

Self-Measurements: On-demand RA



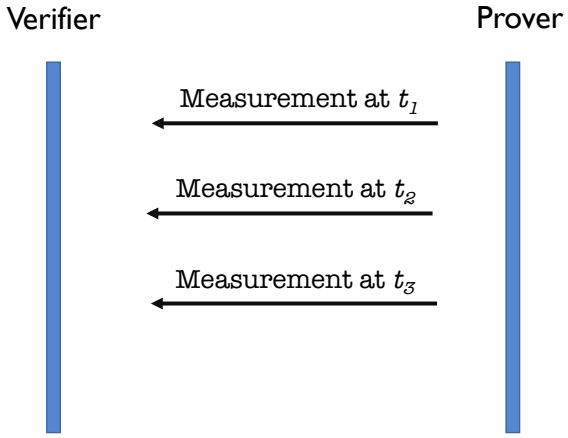
32

Self-Measurements: SeED [WiSec'17]

- Idea: Prover measures itself and immediately sends result to Verifier without any explicit request by the latter
- Unpredictable measurement schedule to protect against transient malware
  - Requires secure real-time clock

33

Self-Measurements: SeED [WiSec'17]



34

Self-Measurements: ERASMUS [DATE'18]

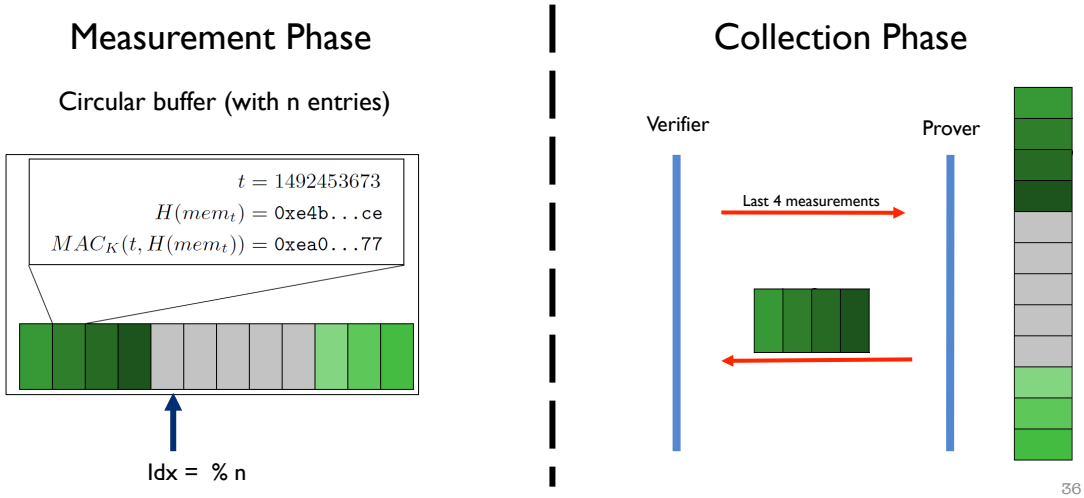
- Similar to SeED, but:

  Instead of sending one result to Verifier after each self-measurement:
  - Accumulates measurements locally
  - Sends **all** (or specified range) to Verifier upon request

- Still requires secure real-time clock to detect transient malware

35

Self-Measurements: ERASMUS [DATE'18]



36

Self-measurements: Feature summary

- No presence of Verifier
  - Suitable for unattended settings
- Unpredictable RA schedule
  - Detects transient malware
- Uninterruptible attestation
  - Detects self-relocating malware
- Additional hardware feature
  - Secure (i.e., reliable, read-only) real-time clock

37

Conclusions

- Tension between secure RA and safety-critical operation
- Surveyed RA techniques attempt to mitigate it by:
  - Memory locking + temporal consistency
  - Shuffled memory traversal
  - Self-measurements
- Identified trade-offs between:
  - Malware detection
  - Temporal consistency guarantees
  - Run-time overhead
  - Hardware requirements

38

Ongoing & Future Work

- Formal verification
- Swarm/group setting
- Extensions (actually, practical applications), e.g.:
  - Secure Reset, Erasure, SW/FW Update

39

\end{document}

% Comments?
% Questions?

40

MITIGATION METHODS	Malware Types Detected		Writable Memory Availability	Consistency Guarantees	Interrupt-ibility	Unattended Setting	Extra HW Require	Run-Time Overhead
	Self-relocating	Transient						
TyTan [DAC'15]								
Temporal Consistency via Memory Locking [ASIACCS'18]								
Shuffled Measurements SMARM [HOST'18]								
Self-measurements SeED, ERASMUS [WiSec'17,DATE'18]								

41