
Karim Eldefrawy and Gene Tsudik

karim.eldefrawy@sri.com
gts@ics.uci.edu

ACM Wisec 2019
Internet-of-Things (IoT)/Embedded Devices
Growing Adoption of IoT/Embedded Devices

https://www.ncta.com/broadband-by-the-numbers
Attacks on (and using) Embedded/IoT Devices
Low-end Embedded/IoT Devices

- Designed for: **Low-Cost, Low-Energy, Small-Size**
- Memory: Program (~32kB) and Data (~2-16 kB)
- Single core CPU (~8-16MHz; 8 or 16 bits)
- Simple Communication Interfaces for IO (a few kbps)
- Example Devices: TI MSP-430, AVR ATMega32 (Arduino)
Remote Attestation (RA)

• A security service for detecting malware on devices

• Interactive challenge-response protocol between:
  • **Verifier**: (computationally powerful) trusted entity
  • **Prover**: potentially infected and remote IoT/embedded device

• **Goal**: to verify internal (memory) state of prover
(1) Challenge

(2) Authenticated memory measurement (via some integrity ensuring function, e.g., HMAC)

(3) Response

(4) Verify response
The Most Important Slide

Developing (Hybrid) RA architectures with solid security guarantees, even if only using simple cryptographic 2 party protocols and for simple devices, is highly non-trivial and difficult.

RA for simple devices is a `perfect’ first candidate for (full) computer-aided formal verification and is within reach of current tools and frameworks.
Attack/Adversary Model

• Remote adversary
  • Exploits vulnerabilities to inject malware

• Local adversary
  • Manipulates communication channel

• Physical adversary
  • Non-Invasive: mounts hardware side-channel attacks
  • Invasive: physically changes hardware
Attack/Adversary Model

- Remote adversary
  - Exploits vulnerabilities to inject malware

- Local adversary
  - Manipulates communication channel

- Physical adversary
  - Non-Invasive: mounts hardware side-channel attacks
  - Invasive: physically changes hardware

This Talk:
Adversary may have full control of the device’s software state
RA Design Classes

1. Hardware-based RA [...] 
   • Dedicated hardware support (e.g., Trusted Platform Module, TPM) 
   • Examples: ARM TrustZone or Intel SGX 
   • Overkill for lower-end IoT and embedded devices

2. Software-based RA [SLS+05, CFSP09, LMP11] 
   • Relies on precise timing measurement 
   • Unrealistic assumptions for IoT and embedded devices (except peripheral/legacy ones)

3. Hybrid RA [EFPT12, ERT17, CERT18, DERT18] 
   • Software/Hardware (SW/HW) co-design 
   • Minimal (additional) hardware requirements 
   • **Best fit for resource constrained IoT and embedded devices**
Example Hybrid RA

• SMART@NDSS’12 [EFPT12]:
  • RA for low-end micro-controllers (e.g., TI MSP430)
  • Established necessary properties for secure RA
Example Hybrid RA

• SMART@NDSS’12 [EFPT12]
  • RA for low-end micro-controllers (e.g., TI MSP430)
  • Established necessary properties for secure RA

• HYDRA@WiSec’17 [ERT17]
  • Emulation of SMART on devices capable of running micro-kernels
  • Based on formally verified seL4 micro-kernel
  • Hardware-assisted secure boot (only hardware feature)
Hybrid RA Security Properties

Key Protection
- Key Access Control
- Key Confidentiality

Secure Remote Attestation

Safe Execution
- Functional Correctness
- Atomicity
- Controlled Invocation
- Attestation Code Immutability

Progression of Research on Hybrid RA

2011
- First Hybrid RA design
  (SMART @ NDSS’12)

2015
- Hybrid RA using the formally verified seL4 microkernel
  (HYDRA @ WISEC’17)

2017
- Hybrid RA protocols for groups of devices (ASIACCS’17),
  temporal consistency of integrity ensuring functions
  (ASIACCS’18)

2018
- First steps towards formal verification of Hybrid RA
  (VRASED @ UsenixSec’19)

2019
- Provable erasure, reset, memory update using VRASED (under review)
- Provable execution using VRASED (under review)

Manual Security Arguments/Proofs and Verification

First Steps in Computer-aided Formal Verification
Why is developing secure RA difficult?
Assumptions and (Architecture) Details Matter

1. Underspecified Interfaces
   • Return address of RA routine could result in indefinite loop

2. Temporal Inconsistency of Inputs to Integrity Ensuring Function (IEF)
   • Computing IEF (e.g., HMAC) takes time
   • Cannot make claims about input to IEF if interrupts allowed during such computation

3. Atomicity of Execution
   • Disabling interrupts is not instantaneous, it may take several clock cycles
Assumptions and (Architecture) Details Matter

1. Underspecified Interfaces
   - Return address of RA routine could result in indefinite loop

2. Temporal Inconsistency of Inputs to Integrity Ensuring Function (IEF)
   - Computing IEF (e.g., HMAC) takes time
   - Cannot make claims about input to IEF if interrupts allowed during such computation

3. Atomicity of Execution
   - Disabling interrupts is not instantaneous, it may take several clock cycles
Assumptions and (Architecture) Details Matter

1. Underspecified Interfaces
   • Return address of RA routine could result in indefinite loop

2. Temporal Inconsistency of Inputs to Integrity Ensuring Function (IEF)
   • Computing IEF (e.g., HMAC) takes time
   • Cannot make claims about input to IEF if interrupts allowed during such computation

3. Atomicity of Execution
   • Disabling interrupts is not instantaneous, it may take several clock cycles
Modeling Temporal Consistency in RA

- Block size of Integrity Ensuring Function (IEF) = memory block size, e.g., 512 bits for HMAC-SHA256
- **IEF** is a sequential function: process each block once and in order
- Content of blocks may change during execution of **IEF**

$M_i := \text{Content of } i^\text{th block}$

$M_i^t := M_i \text{ at time } t$

$M^t := M \text{ at time } t$
Modeling Temporal Consistency in RA

- Block size of Integrity Ensuring Function (IEF) = memory block size, e.g., 512 bits for HMAC-SHA256
- IEF is a sequential function: process each block once and in order
- Content of blocks may change during execution of IEF
- However, fetching $M_i$ is an atomic operation

Output $R$ of IEF is consistent with $M$ iff: $R = IEF(M^t)$
Modeling Temporal Consistency in RA

- Block size of Integrity Ensuring Function (IEF) = memory block size, e.g., 512 bits for HMAC-SHA256
- **IEF** is a sequential function: process each block once and in order
- Content of blocks may change during execution of **IEF**
- However, fetching \( M_i \) is an atomic operation

\[ M_i := \text{Content of } i^{\text{th}} \text{ block} \]
\[ M_i^t := M_i \text{ at time } t \]
\[ M^t := M \text{ at time } t \]

**Output** \( R \) of **IEF** is consistent with \( M \) iff:
\[ R = IEF(M^t) \]
Is computer-aided formal verification of RA feasible?
First Steps Towards Computer-aided Formal Verification of RA

- Verifiable Remote Attestation for Simple Embedded Devices (VRASED)
- Instantiates a (mechanically) verified SMART-like hybrid (HW/SW) RA co-design aimed at low-end embedded devices
- Accepted (Minor Revisions) at Usenix Security 2019
Overview of VRASED

PC = Program Counter, R/W<sub>en</sub> = Read/Write enable, D<sub>addr</sub> = Data address, DMA<sub>en/addr</sub> = Direct Memory Access enable/address
Overview of VRASED

- Verified Implementation of Integrity-Ensuring Function (HMAC) stored in *read only memory* (*SW-Att*): Immutability + SW correctness
  - Malware cannot modify the *SW-Att*

PC = Program Counter, R/W\textsubscript{en} = Read/Write\textsubscript{enable}, D\textsubscript{addr} = Data\textsubscript{address}, DMA\textsubscript{en/addr} = Direct Memory Access\textsubscript{enable/address}
Recall: Hybrid RA Security Properties
Overview of VRASED

- Verified Implementation of Integrity-Ensuring Function (HMAC) stored in read only memory (SW-Att): Immutability + SW correctness
  - Malware can not modify the SW-Att

- Does SW-Att execute properly?
  - HW-Mod enforces proper invocation and atomicity
Recall: Hybrid RA Security Properties

- Key Protection
  - Key Access Control
  - Key Confidentiality

- Secure Remote Attestation
  - Functional Correctness

- Safe Execution
  - Atomicity
  - Controlled Invocation
  - Attestation Code Immutability
Overview of VRASED

- Verified Implementation of Integrity-Ensuring Function (HMAC) stored in read only memory (SW-Att): Immutability + SW correctness
  - Malware can not modify the SW-Att

- Does SW-Att execute properly?
  - HW-Mod enforces proper invocation and atomicity

- Can Malware learn something it should not from SW-Att execution?
  - HW-Mod makes sure it cannot
  - Access control to the key and to memory used by SW-Att

PC = Program Counter, R/W_en = Read/Write enable, D_addr = Data address, DMA_en/addr = Direct Memory Access enable/address
Recall: Hybrid RA Security Properties

- Secure Remote Attestation
  - Key Protection
    - Key Access Control
    - Key Confidentiality
  - Safe Execution
    - Functional Correctness
    - Atomicity
    - Controlled Invocation
    - Attestation Code Immutability
Details of Verification

1. Rely on a verified HMAC (from HACL*) implementation for SW-Att
2. Model RA security properties as Linear Temporal Logic (LTL) specs
3. Design HW-Mod as a set of FSMs and use a model-checker to verify its conformance to LTL Specs
4. Prove in LTL that the combination of SW-Att + HW-Mod properties imply “Secure RA”
Overview of Linear Temporal Logic (LTL)

- $X\phi$ – next $\phi$: holds if $\phi$ is true at the next system state.
- $F\phi$ – Future $\phi$: holds if there exists a future state where $\phi$ is true.
- $G\phi$ – Globally $\phi$: holds if for all future states $\phi$ is true.
- $\phi U \psi$ – $\phi$ Until $\psi$: holds if there is a future state where $\psi$ holds and $\phi$ holds for all states prior to that.
Key Access Control in LTL

\[ G: \neg (PC \in CR) \land Ren \land (Daddr \in KR) \rightarrow \text{reset} \]
Key Access Control in LTL

\[ G: \{ \neg (PC \in CR) \land Ren \land (Daddr \in KR) \rightarrow \text{reset} \} \]

- Not attestation code
- Trying to read secret Key
Key Access Control in LTL

\[ G: \neg (PC \in CR) \land Ren \land (Daddr \in KR) \rightarrow \text{reset} \]
Atomicity and Controlled Invocation in LTL

If executing an instruction from attestation code either reach end or reset

\[
G : \{ \\
\neg reset \land (PC \in CR) \land \neg(X(PC) \in CR) \rightarrow \\
PC = CR_{max} \lor X(reset) \}
\]

Can only start from first instruction in attestation code or reset
If SW-Att is stored in memory $[CR_{\text{min}}, CR_{\text{max}}]$ and a challenge is received and the key is stored in the correct memory location KR (only accessible to SW-Att) then ...

**Definition 3.**

$$
G: \{ \\
PC = CR_{\text{min}} \land MR = \text{Chal} \land \\
[\neg \text{reset} \land CR = \text{SW-Att} \land KR = \mathcal{K} \land AR = M) \cup PC = CR_{\text{max}}] \\
\rightarrow F: [PC = CR_{\text{max}} \land MR = \text{HMAC(KDF(\mathcal{K}, \text{Chal}), M)}] \} \\
$$

where $M$ is any arbitrary value for AR.
If SW-Att is stored in memory $[CR_{\text{min}}, CR_{\text{max}}]$ and a challenge is received and the key is stored in the correct memory location KR (only accessible to SW-Att) then ....

all these properties hold till

$PC = CR_{\text{max}}$

...
If SW-Att is stored in memory $[CR_{\text{min}}, CR_{\text{max}}]$ and a challenge is received and the key is stored in the correct memory location KR (only accessible to SW-Att) then .... all these properties hold till $PC = CR_{\text{max}}$ ...

reach execution of last instruction of SW-Att and the value in MR is the IEF/HMAC over the memory (suing the correct key and challenge)
(Déjà Vu) The Most Important Slide

Developing (Hybrid) RA architectures with solid security guarantees, even if only using simple cryptographic 2 party protocols and for simple devices, is highly non-trivial and difficult.

RA for simple devices is a "perfect" first candidate for (full) computer-aided formal verification and is within reach of current tools and frameworks.
Future Work

• Formally specify and mechanically verify both SW and HW within one interactive proof assistant/system, e.g., Coq, Isabelle, or others. LTL cannot do this.

• Verifiably synthesize software and hardware implementations of the design(s)

• Formally specify and mechanically verify protocols for group settings

• Newer processor architectures, e.g., RISC-V

• More experiments and applications, e.g., verifiable erasure, resetting, updating, authenticated sensors that “cannot lie”

• More complex devices with full OS
Questions?
References