www.crysys.hu

T-RAID – TEE-based Remote Attestation for IoT Devices

Roland Nagy, Márton Bak, Dorottya Papp, Levente Buttyán
CrySyS Lab, BME
www.crysys.hu

EuroCyberSec 2021 Workshop, Nice, Oct 25-26, 2021

Introduction

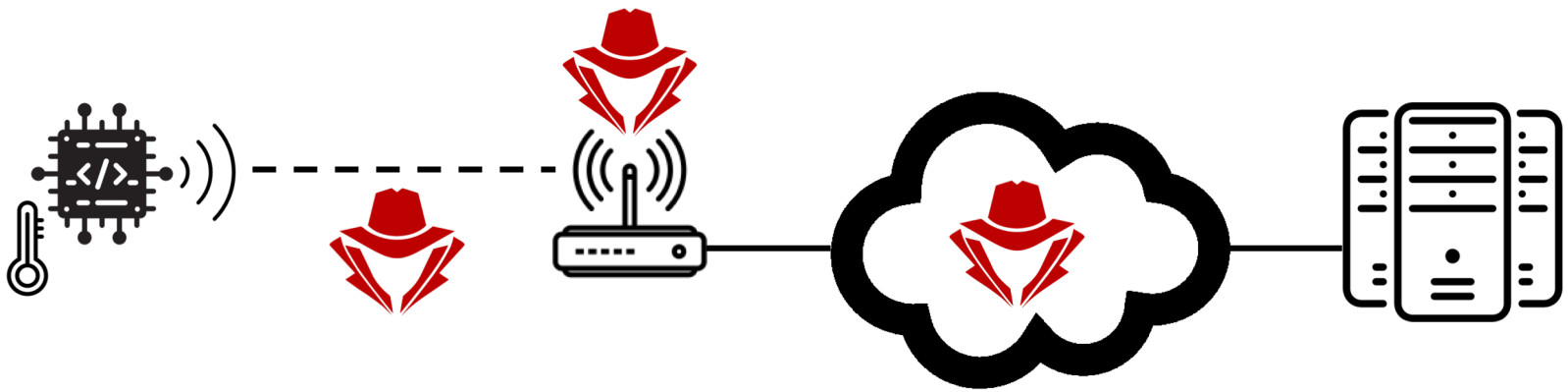
- Imagine an IoT system, e.g., a smart home with sensors, actuators, and a cloud-based back-end system
- The back-end receives a message:

```
sensor id: 80a56ca0ff38
type: temperature
value: 22.5
timestamp: 1430904085
```

- What does this mean?

Introduction

The issue of Man-in-the-Middle attacks…



```
sensor id: 80a56ca0ff38
type: temperature
value: 22.5
timestamp: 1430904085
```

→ modify / inject / spoof →

```
sensor id: 80a56ca01238
type: temperature
value: 16.5
timestamp: 1430904196
```

Introduction

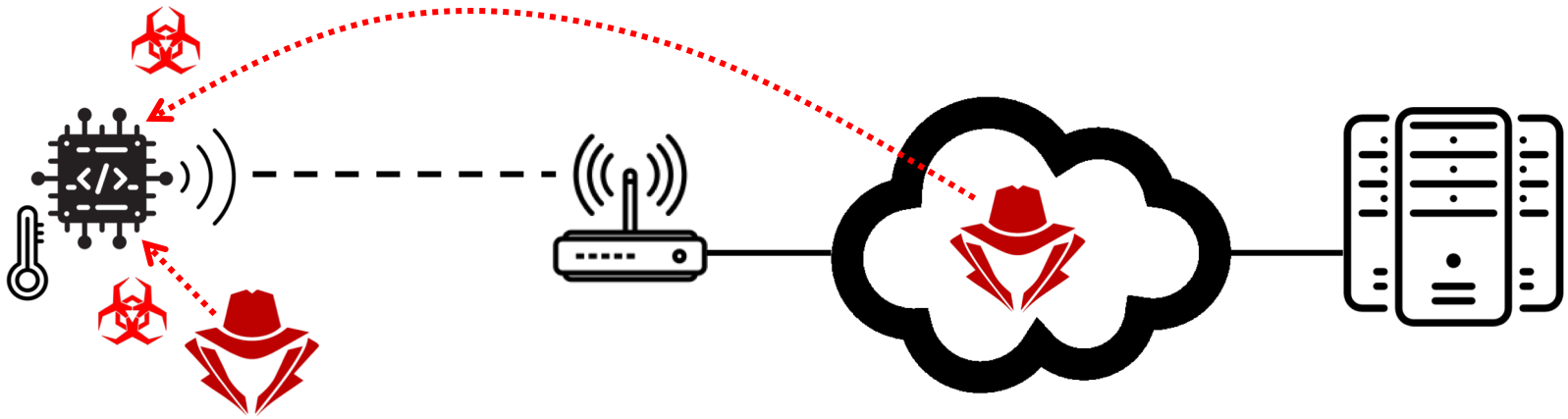
- The back-end receives a message **over an authenticated and integrity-protected end-to-end connection**:

```
sensor id: 80a56ca0ff38
type: temperature
value: 22.5
timestamp: 1430904085
```

- What does this mean?

Introduction

The issue of device compromise …



```
sensor id: 80a56ca0ff38
type: temperature
value: 22.5
timestamp: 1430904085
```

modify
inject
(spoof)

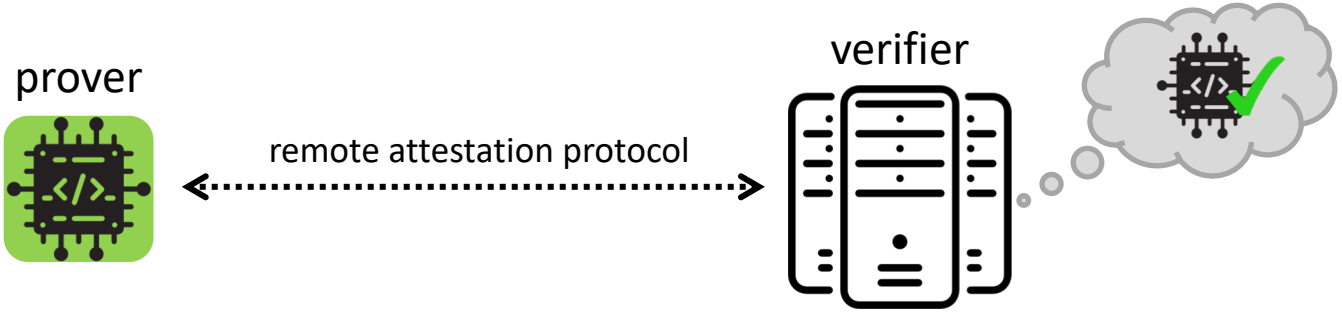
secure channel

```
sensor id: 80a56ca0ff38
type: temperature
value: 16.5
timestamp: 1430904196
```

Introduction

- Can the IoT device prove its current malware-free state to the remote back-end using some protocol?
- The main challenge is that if the device is compromised, then it can lie about its state
  - e.g., the naïve solution of sending the hash of the current memory content of the prover to the verifier does not work
- How can we ensure that the protocol is successfully completed only if the IoT device is intact?

Remote attestation



- Attestation is a process whereby a trusted verifier checks the state of an untrusted prover
- Remote attestation is attestation performed over a network
- The prover may be an embedded IoT device
- The goal of attestation is to prove the malware-free state of the IoT device
- This is achieved by executing a protocol in which the verifier probes the prover and the response of the prover convinces the verifier

Approaches to remote attestation

- Hardware-based
  - Relies on a secure co-processor (e.g., TPM chip)
  - The co-processor can produce a digitally signed summary (e.g., hash) of the hardware and software state of the system
  - The signing key is kept in and protected by the co-processor
  - This approach may be too expensive for IoT devices

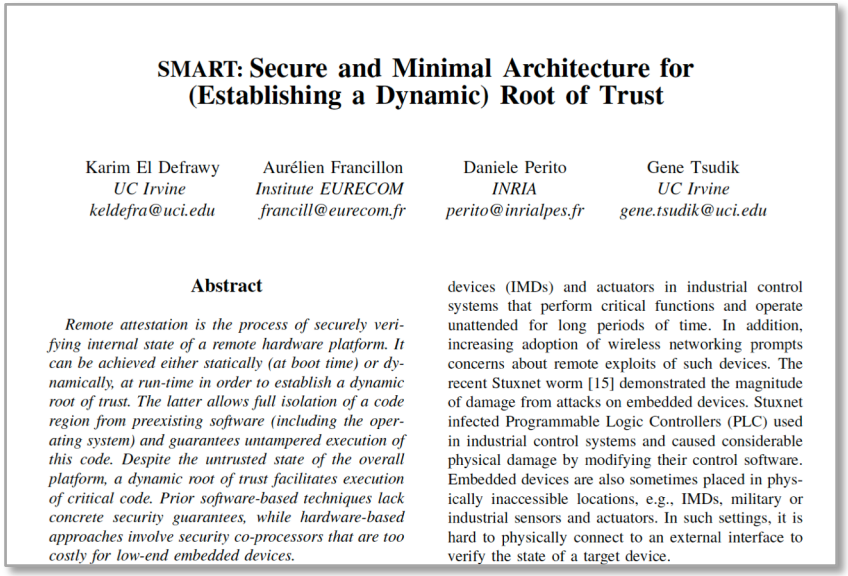
- Software-based
  - Does not rely on any additional hardware
  - The prover typically runs an optimized checksum function which traverses memory locations in a pseudo-random manner (seeded by the verifier challenge)
  - The verifier checks the correctness of the prover's response **and the response time**
    - » Hiding a malware in memory would result in a longer checksum computation time
  - In practice, this approach does not really work over a network
    - » Network jitter makes it practically impossible to reliably measure the checksum computation time
    - » A compromised prover can actually delegate checksum computation to a much faster attacker device, which cannot be detected by a remote verifier

Approaches to remote attestation

- Hybrid approach
  - Largely software-based, but also uses minimal hardware support
  - Examples:
    - » El Defrawy *et al*, **SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust**, NDSS 2012
    - » El Defrawy *et al*, **HYDRA: HYbrid Design for Remote Attestation (Using a Formally Verified Microkernel)**, WiSec 2017

<u>SMART</u>

- ROM-based checksum routine (cannot be modified)
- Checksum is authenticated by a secret key
- Secret key is kept in memory that can only be accessed from ROM-based code
  - » memory access hardware logic verifies that the instruction pointer is in the ROM region when the secret key is being accessed



SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust

Karim El Defrawy, *UC Irvine*, *keldefra@uci.edu*
Aurélien Francillon, *Institute EURECOM*, *francill@eurecom.fr*
Daniele Perito, *INRIA*, *perito@inrialpes.fr*
Gene Tsudik, *UC Irvine*, *gene.tsudik@uci.edu*

Abstract

Remote attestation is the process of securely verifying internal state of a remote hardware platform. It can be achieved either statically (at boot time) or dynamically, at run-time in order to establish a dynamic root of trust. The latter allows full isolation of a code region from preexisting software (including the operating system) and guarantees untampered execution of this code. Despite the untrusted state of the overall platform, a dynamic root of trust facilitates execution of critical code. Prior software-based techniques lack concrete security guarantees, while hardware-based approaches involve security co-processors that are too costly for low-end embedded devices.

devices (IMDs) and actuators in industrial control systems that perform critical functions and operate unattended for long periods of time. In addition, increasing adoption of wireless networking prompts concerns about remote exploits of such devices. The recent Stuxnet worm [15] demonstrated the magnitude of damage from attacks on embedded devices. Stuxnet infected Programmable Logic Controllers (PLC) used in industrial control systems and caused considerable physical damage by modifying their control software. Embedded devices are also sometimes placed in physically inaccessible locations, e.g., IMDs, military or industrial sensors and actuators. In such settings, it is hard to physically connect to an external interface to verify the state of a target device.

Minimal requirements on hardware?

Francillon *et al*, **A Minimalist Approach to Remote Attestation**, Design, Automation & Test in Europe, 2014

1. Custom hardware to enforce exclusive access to a secret key
2. Reliable and secure memory erasure
3. Read-only-memory (ROM)
4. Enable-interrupts and atomic disable-interrupts instructions
5. Custom hardware to enforce that the attestation routine is invokable only at the first instruction
6. Secure reset mechanism

HYDRA

Uses a formally verified seL4 microkernel to obtain the above properties

1. A privileged process handles the key
2. Strict memory separation
3. Isolated process memory and code integrity checks + secure boot
4. Prioritized interrupt handling
5. OS support

HYDRA: HYbrid Design for Remote Attestation (Using a Formally Verified Microkernel)

Karim Eldefrawy*
Information and Systems Sciences Lab
HRL Laboratories
keldefrawy@hrl.com

Norrathep Rattanavipanon
Computer Science Department
University of California, Irvine
nrattana@uci.edu

Gene Tsudik
Computer Science Department
University of California, Irvine
gene.tsudik@uci.edu

ABSTRACT

Remote Attestation (RA) allows a trusted entity (*verifier*) to securely measure internal state of a remote untrusted hardware platform (*prover*). RA can be used to establish a static or dynamic root of trust in embedded and cyber-physical systems. It can also be used as a building block for other security services and primitives, such as software updates and patches, verifiable deletion and memory resetting. There are three major types of RA designs: *hardware-based*, *software-based*, and *hybrid*, each with its own set of benefits and drawbacks.

This paper presents the first hybrid RA design – called HYDRA – that builds upon formally verified software components that ensure memory isolation and protection, as well as enforce access control to memory and other resources. HYDRA obtains these properties by using the formally verified seL4 microkernel. (Until now, this was only attainable with purely hardware-based designs.) Using seL4 imposes fewer hardware requirements on the underlying microprocessor that were previously analog (or at least not connected) brings many obvious benefits. However, it also expands the attack surface and turns these newly computerized gadgets into natural and attractive attack targets.

Remote Attestation (RA) is the process whereby a trusted entity called "verifier" securely probes internal state of a remote and untrusted hardware platform, called "prover." RA can be used to establish a static or dynamic root of trust in ES, CPS and IoT devices. Also, RA can be used as a foundation for constructing more specialized security services, e.g., software updates, verifiable deletion and memory resetting. There are three main classes of RA designs: *hardware-based*, *software-based*, and *hybrid* (blending hardware and software). Each class has its own advantages and limitations. This paper introduces *the first* hybrid RA design – called HYDRA – based upon formally verified components to provide memory isolation and protection guarantees. Our main rationale is that designing RA techniques based upon such components increases confidence

Our approach

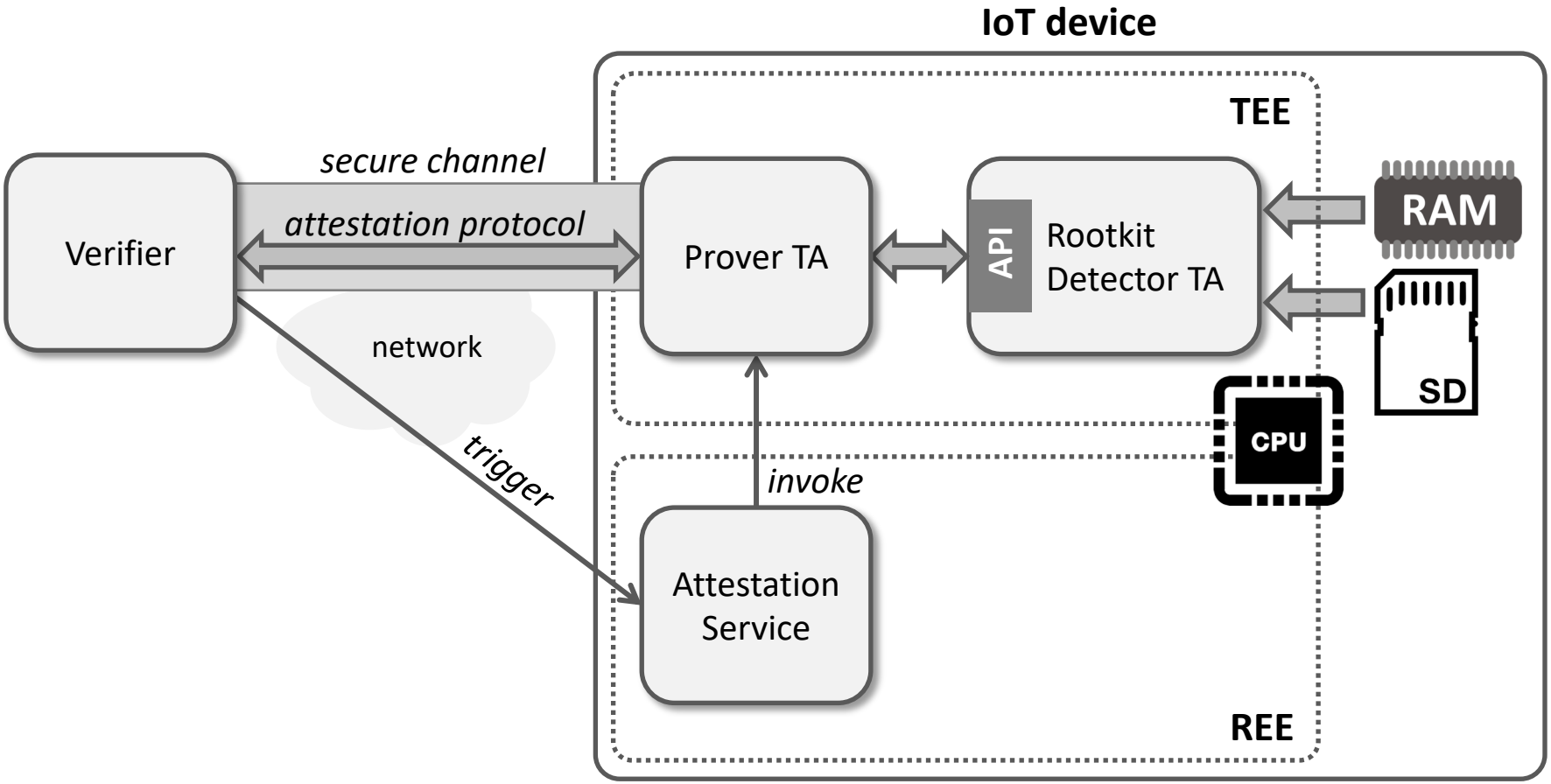
Obtain the required properties by relying on a TEE (Trusted Execution Environment)

Note that proper implementation of a TEE needs some hardware support

- » separation of a secure and a normal CPU mode
- » hardware enforced access control to the memory

1. A trusted application (TA) handles a private attestation key and uses that to setup a secure communication channel with the verifier
2. TEE-enforced memory separation shields the private key in memory from other processes
3. TEE-based integrity protection of TAs prevents their illegitimate modification + secure boot
4. Possibility of disabling interrupts in the TEE ensures uninterruptable execution of the attestation TA
5. TEE-based invocation mechanism enforces that the execution of a TA always begins at its entry point

T-RAID architecture



T-RAID implementation

- Platform → ARM with TrustZone support (e.g., iMX6)
- REE → Linux
- TEE → OP-TEE
- Cryptographic functions → mbedTLS library

- <u>Known weakness:</u>
  - file operations in OP-TEE are delegated to the REE side, where they are actually interruptable
    - » File integrity checks are not reliable
    - » Persistent malware may escape detection
  - on multi-core processors, other processes may run in parallel to our TAs on different cores
    - » Some of our integrity checks may not be reliable
    - » It is not clear yet, whether it is technically possible to exploit this in practice

Secure channel establishment

- One can use TLS if…
  - it is available within the TEE
  - resource constraints of the IoT device permit it

- As OP-TEE does not support TLS protected sockets inside the TEE, we implemented our own lightweight secure channel protocol
  - ECDH key exchange with ECDSA signed messages
  - Key derivation with PBKDF2
    - » 32-byte AES key
    - » 32-byte HMAC key
  - AES-CBC encryption with PKCS#7 padding
  - HMAC-SHA256 message authentication (on encrypted messages)
  - Replay protection with sequence numbering messages

Attestation protocol

- Contains a single request – response exchange
- Request sent by the verifier
  - May contain multiple lines
  - Each line triggers the call of a specific integrity check function of the Rootkit Detector TA via its API
- Response sent by the prover
  - Contains the results of the integrity check functions called
  - Integrity check functions may return
    - » a status code (e.g., 0 for a successful check and 1 for a failure)
    - » a hash value (e.g., hash of the kernel's text segment or recursive hash of some part of the file system)
    - » a list of process IDs and process names extracted from various kernel data structures (e.g., process list or process tree)
- The verifier should be able to check the responses
  - It stores expected hash values
  - In case of file system checks, it computes the expected result on a mirror of the file system of the IoT device
  - It can compare the received list of process names to a white list

Integrity checks

R. Nagy, K. Németh, D. Papp, and L. Buttyán, **Rootkit Detection on Embedded IoT Devices**, Acta Cybernetica, 2021.

- Check processes
  - Extracts process related information from different kernel data structures (process list, process tree, run queues)
  - Returns a list of process IDs and a list of process names
- Memory integrity check
  - Computes and returns the hash of the system call table and the kernel's text segment
- Network checks
  - Checks if all function pointers in the network stack point inside the text segment of the kernel
  - Returns success or failure
- File integrity checks
  - Recursively computes and returns the hash of the files in a specified folder
  - A black list could also be supplied that contains files (their names) that are not included in the hash computation

Attestation query language

- A line in an attestation request may contain the name of the integrity check function to be called and its parameters

  Examples:

  - HASH_DIR --recursive /opt/
  - HASH_FILE /etc/passwd
  - HASH_KERNEL_TEXT
  - HASH_SYSCALL
  - NETFILTER_HOOKS_CHECK
  - PROCLIST

Conclusions

- We proposed T-RAID, a TEE-based remote attestation method for embedded IoT devices
- T-RAID is conceptually simple
  - Assuming integrity checks to prove rootkit freedom are available
- Limitations of actual TEE implementations may introduce weaknesses in the implementation of T-RAID
- The “multi-core problem” needs further investigation

Acknowledgement

The presented work was carried out within the SETIT Project, which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme (project no. 2018-1.2.1-NKP-2018-00004).

NEMZETI KUTATÁSI, FEJLESZTÉSI ÉS INNOVÁCIÓS HIVATAL

AZ INNOVÁCIÓ LENDÜLETE

AZ NKFI ALAPBÓL MEGVALÓSULÓ PROJEKT