Research Project Proposal: Interruptible Remote Attestation

Davide Li Calsi
davide.li@mail.polimi.it
CSE Track

POLITECNICO MILANO 1863

HONOURS PROGRAMME HP-SR in Information Technology

Agenda

- Remote Attestation
- RA and Interrupts
- Performance Counters for Malware Detection
- A new approach to interruptible RA

Remote Attestation

What is it? How does it work?

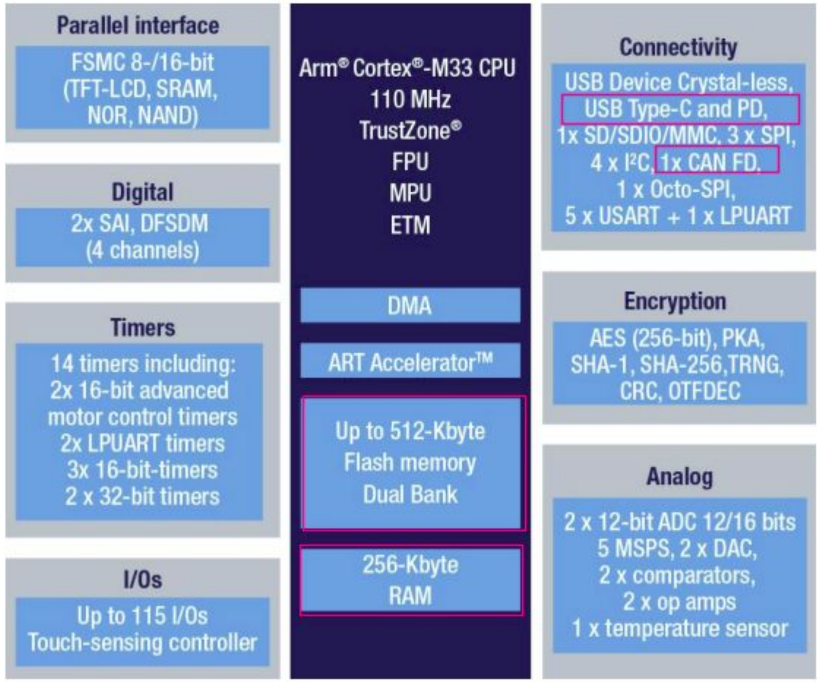
Low-end MCUs and security

Security is a notoriously **expensive property** from a computational standpoint.

What if you are dealing with a heavily constrained device?

Low-end Microcontrollers: scarce computational power, few hundreds of kB of RAM, lack common hardware protection.

ST-Nucleo L-552 board by STM32



Parallel interface
FSMC 8-/16-bit (TFT-LCD, SRAM, NOR, NAND)
Digital
2x SAI, DFSDM (4 channels)
Timers
14 timers including: 2x 16-bit advanced motor control timers 2x LPUART timers 3x 16-bit-timers 2 x 32-bit timers
I/Os
Up to 115 I/Os Touch-sensing controller
Arm® Cortex®-M33 CPU
110 MHz
TrustZone®
FPU
MPU
ETM
DMA
ART Accelerator™
Up to 512-Kbyte Flash memory Dual Bank
256-Kbyte RAM
Connectivity
USB Device Crystal-less, USB Type-C and PD, 1x SD/SDIO/MMC, 3 x SPI, 4 x I²C, 1x CAN FD, 1 x Octo-SPI, 5 x USART + 1 x LPUART
Encryption
AES (256-bit), PKA, SHA-1, SHA-256,TRNG, CRC, OTFDEC
Analog
2 x 12-bit ADC 12/16 bits 5 MSPS, 2 x DAC, 2 x comparators, 2 x op amps 1 x temperature sensor

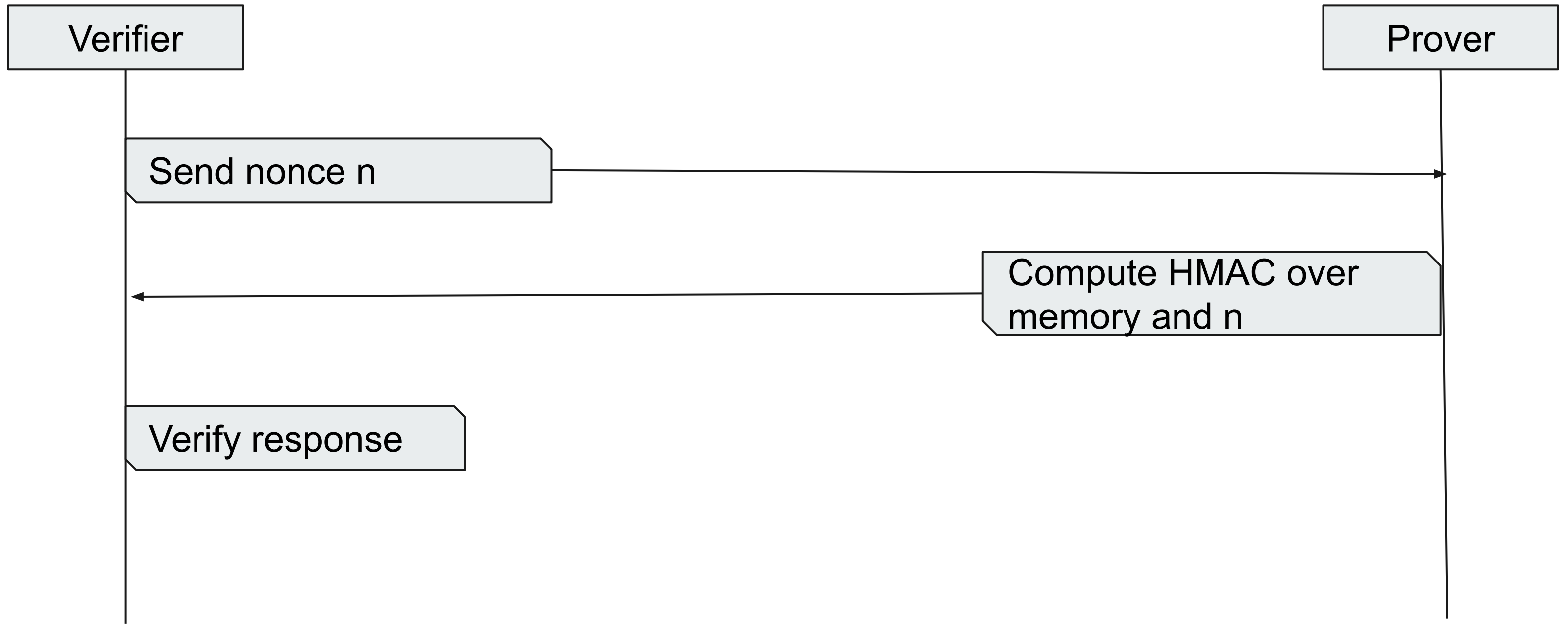
Remote Attestation

A protocol through which a **remote Verifier** can attest the integrity of a **target Prover.**

Allows to start some incident response when compromised devices are spotted.

It has been a very active research field for decades.

Remote Attestation (2)



Remote Attestation and MCUs - Real scenario

Imagine you have a low-end MCU employed in an energy production facility.

What if some attacker decides to compromise it?

Sadly, this scenario has become much more realistic, given the current international situation…

We could use an effective RA protocol here…

50 shades of Remote Attestation

Several classifications exist, depending on what you attest and how.

Static vs Dynamic: attestprogram memory only **vs.** attest data memory and runtime integrity

Software vs Hardware vs Hybrid: software protection only **vs** hardware protection mechanisms **vs** add minimal hardware + software checks

Swarm attestation: attest entire groups of devices faster than when you are attesting each device at a time.

Software-based RA

It uses software-based techniques such that tampering with the Attestation Routine introduces a **significant computational delay**.

If the Verifier measures a significant delay in the response the Prover is compromised.

It only works if the Prover and the Verifier have a single-hop connection.

Communication delays in the network can lead to misclassification.

PROS: simple, flexible, cheap

CONS: relies on strong assumptions, vulnerable to compression and ROP attacks

Hardware-based RA

It uses **hardware-based protection**. Well-known examples are **Intel SGX and ARM Trustzone.**

It provides a **strong and reliable** protection, but it comes at a cost.

Hardware is not suitable for remote updates.

PROS: **strong guarantees, hard to tamper**

CONS: **expensive, lack of flexibility**

Example: ARM Trustzone

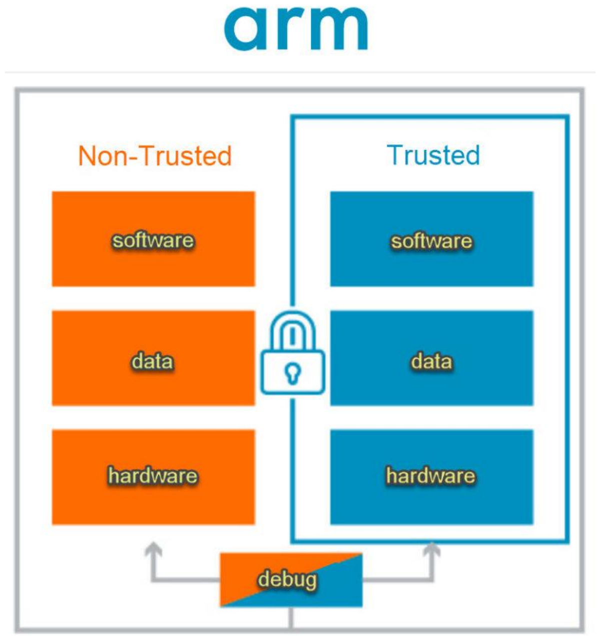


Image by ARM

Hybrid RA

A **hardware-software codesign.** You aim to introduce minimal external hardware components, which is also supported by software checks.

It has been the most active research path lately.

PROS: contained production costs, good balance

CONS: development cost for further hardware

How can the Verifier check the response?

The verification phase depends on **what you are attesting.**

Static attestation: Verifier knows in advance which programs should run on the Prover. Pre-compute the expected attestation response and compare it with the receive one.

Dynamic attestation: requires a more sophisticated analysis on the runtime environment, the Control Flow Graph, ec...

RA and Interrupts

Why it is a problem?

Firmware modification attacks

Leverage flaws in embedded software in order to **modify the device's firmware.**

You can modify the firmware to run pretty much whatever you want.

Example: HP LaserJet Printer, exploit based on a flaw in the HP remote firmware update functionality. More here

Interrupts in RA

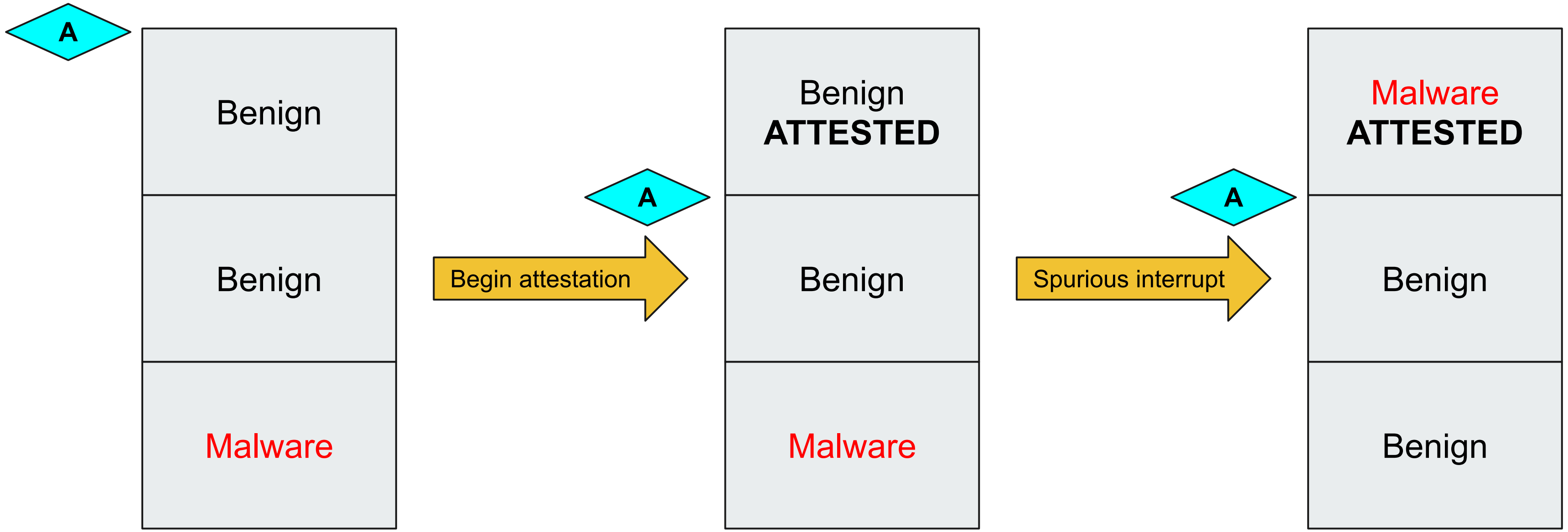
Most current RA techniques require interrupts to be disabled.

Prevents malware that is already in the Flash memory from modifying memory at attestation time, in order to escape detection.

Self-relocating malware: malicious code that erases itself and moves to a location that was already attested.

Transient malware: malware that self-erases to later re-infect the system.

Interrupts in RA - Relocation



The main issue

Interrupts are a **key feature** that should not be disabled for too long.

Time-critical applications need interrupts to be enabled at all time.

Attesting few MegaBytes of memory takes hundreds of ms on average. That is simply not acceptable.

Honorable mentions that allow interrupts

TrustLite: it uses a secure exception engine to handle interrupts

TyTan: attests one process at a time. All processes except for the one that is being attested can interrupt the Attestation Routine

Both schemes allow interrupts during attestation, but are vulnerable to roving malware.

Shuffled measurements

Determine a random permutation that dictates in which order these blocks must be attested.

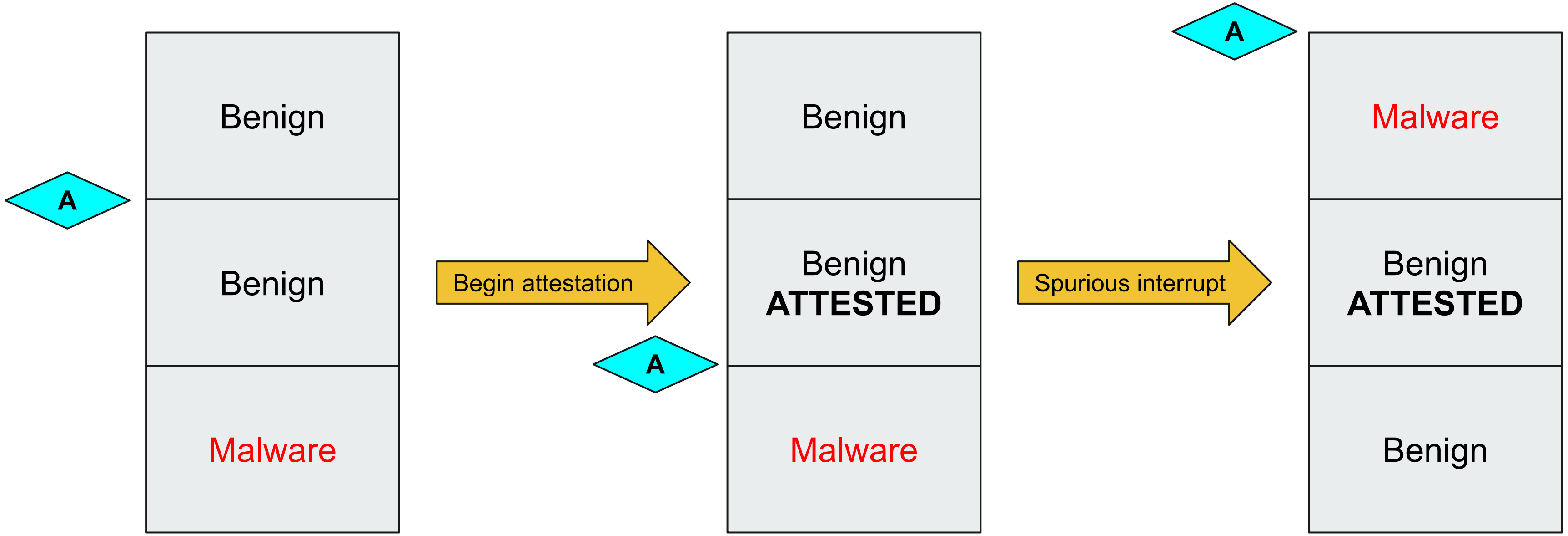
Only probabilistic guarantees, with probabilities depending on the attacker’s knowledge (63%).

To compensate for this, you should repeat attestation multiple times in a row.

Partial interruption: you cannot interrupt while the attestation routine is attesting a single block.

An example

Permutation: 2 1 3



Memory Locks

Locking a memory area = make it read only.

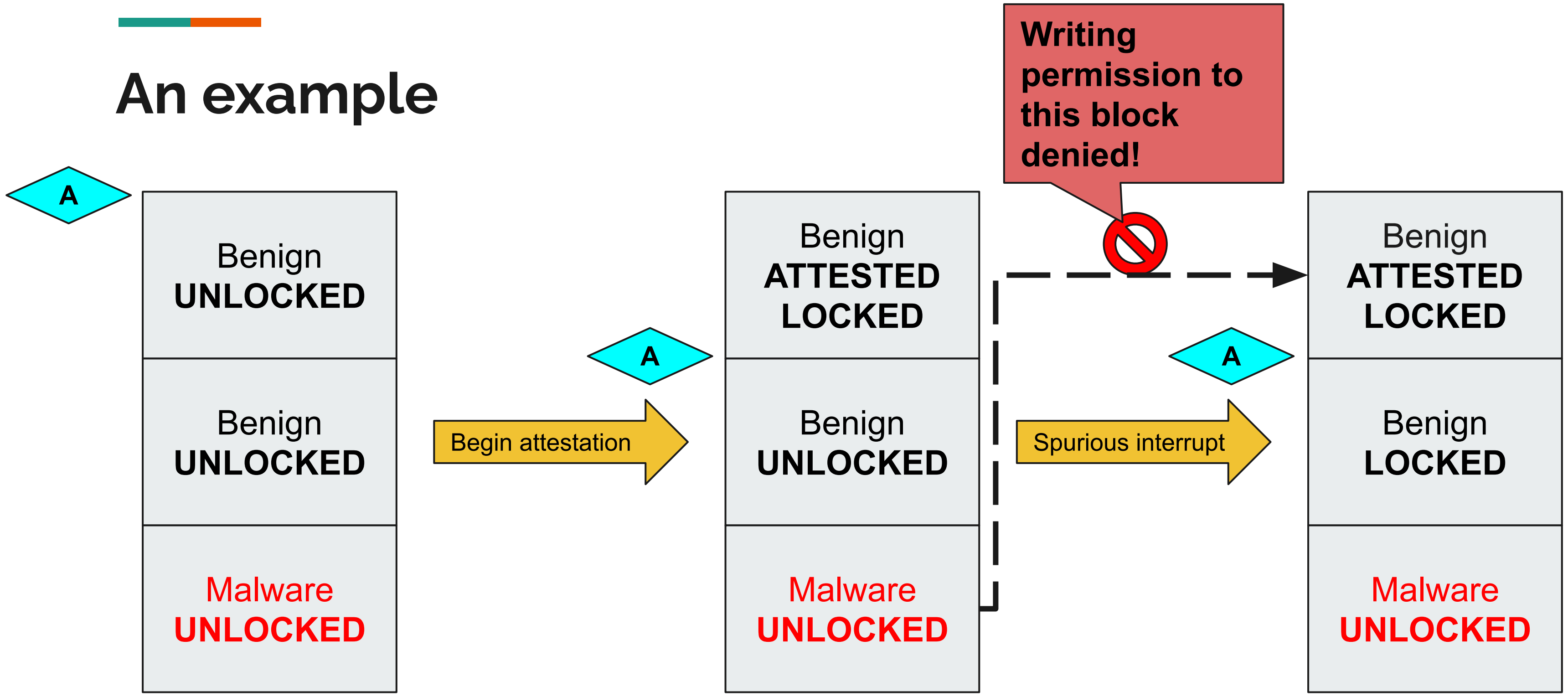
By appropriately locking memory at attestation time, you can allow interrupts.

Many possible implementations, depending on what you lock.

Shortcoming: part of the memory is still not writable!

Shortcoming 2: its implementation is based on system calls, it requires an underlying microkernel. Not suitable for low end devices.

An example


A
Benign UNLOCKED
Benign UNLOCKED
Malware UNLOCKED
Begin attestation
A
Benign ATTESTED LOCKED
Benign UNLOCKED
Malware UNLOCKED
Writing permission to this block denied!
Spurious interrupt
A
Benign ATTESTED LOCKED
Benign LOCKED
Malware UNLOCKED

What is missing?

A technique to make **RA interruptible** without giving up on security.

Should work on low-end microcontrollers: negligible **computational overhead.**

Higher guarantees compared to Shuffled Measurements RA.

Flexible and general enough to be extended to several use-cases.

Performance Counters for Malware Detection

An interesting concept

Malware detection

Is there a better way to detect malicious interrupts?

Reasonable assumption: **infected devices behave differently compared to uncompromised devices.**

Hardware Performance Counters can measure this difference in the behavior.

We conducted a parallel literature review on **malware detection using Performance Counters.**

Which counters?

Performance Counters are natively present in many renown architectures.

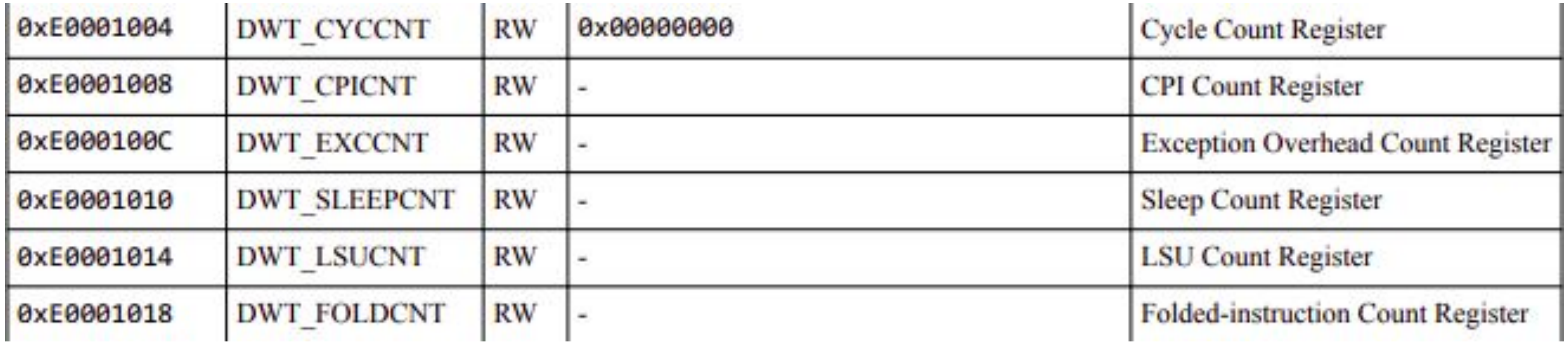
Several CPUs are equipped with a **Power Management Unit (PMU)** or a **DWT(Data Watchpoint and Trace).**

Low-level counters that count the occurrences of architectural events, such as:

- branches
- cache hits/misses
- CPI
- clock cycles spent doing something meaningful

DWT counters

These are the 6 vanilla counters in a Cortex-M-33 microcontroller



0xE0001004	DWT_CYCCNT	RW	0x00000000	Cycle Count Register
0xE0001008	DWT_CPICNT	RW	-	CPI Count Register
0xE000100C	DWT_EXCCNT	RW	-	Exception Overhead Count Register
0xE0001010	DWT_SLEEPCNT	RW	-	Sleep Count Register
0xE0001014	DWT_LSUCNT	RW	-	LSU Count Register
0xE0001018	DWT_FOLDCNT	RW	-	Folded-instruction Count Register

Image by ARM

MTB

MTB is an optional feature of some Cortex-M microcontrollers.

It writes every non-sequential update of the Program Counter to a Memory Buffer.

We could use that extra information to generate **additional counters.**

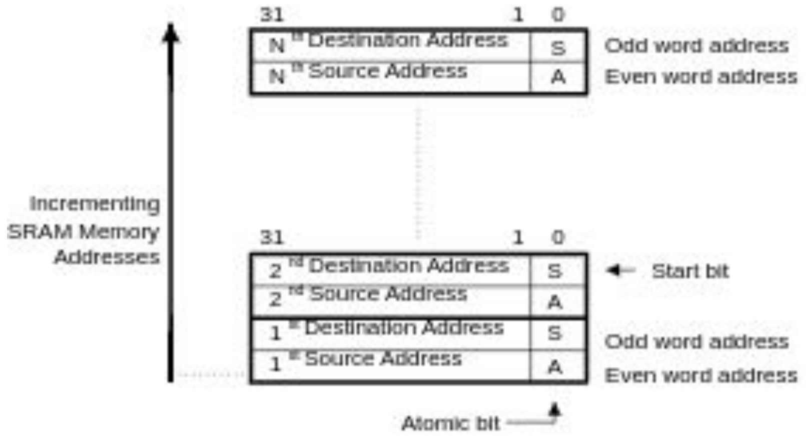


Image by ARM

Classify the result

Some approaches use a Database of benign counters values.

Modern approaches rely on some **Machine Learning model** to classify the result.

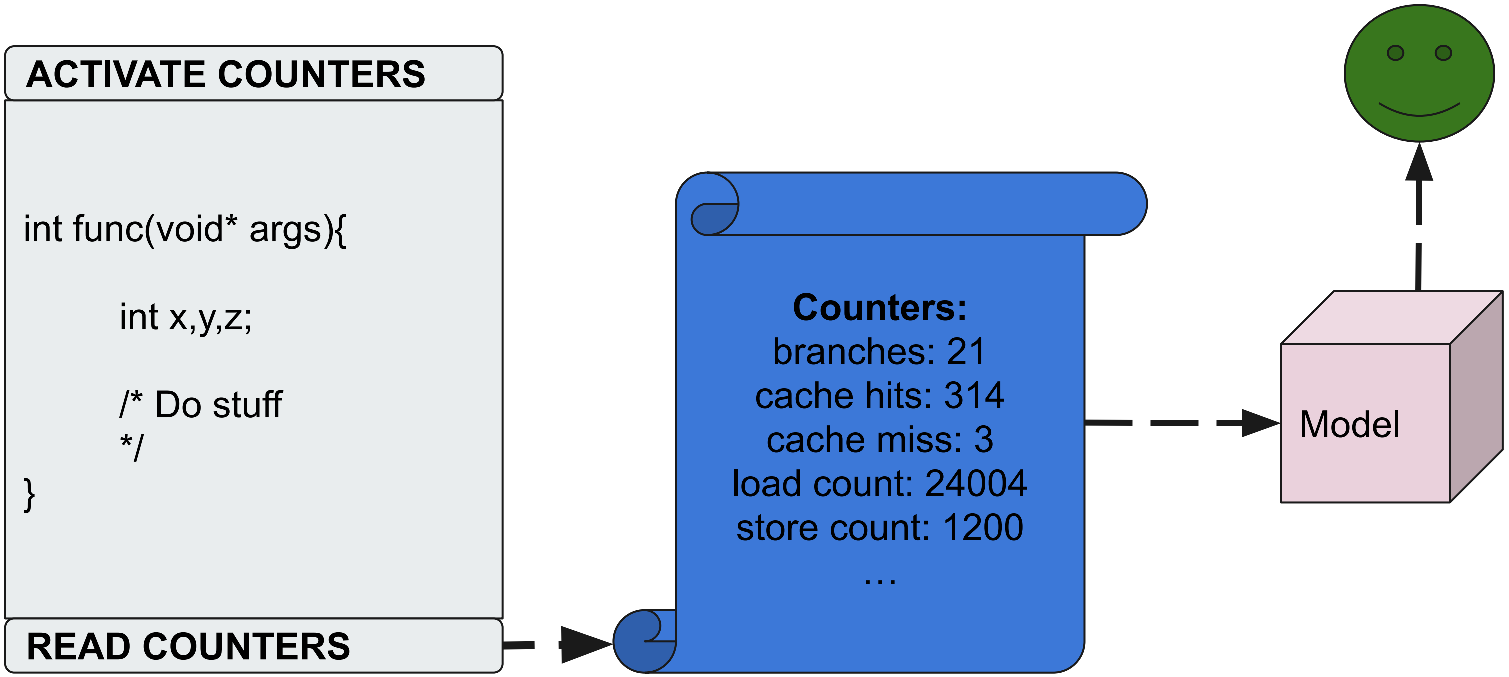
Across the years, several models were evaluated, from Decision Trees to SVM and Neural Networks.

Two phases

Offline phase: pre-deployment phase that consists in running several tests. Collecting enough data to train a Machine Learning model.

Online phase: feed the measured counters value to the pre-trained model, determine which type of activity you have measured

Example - Benign



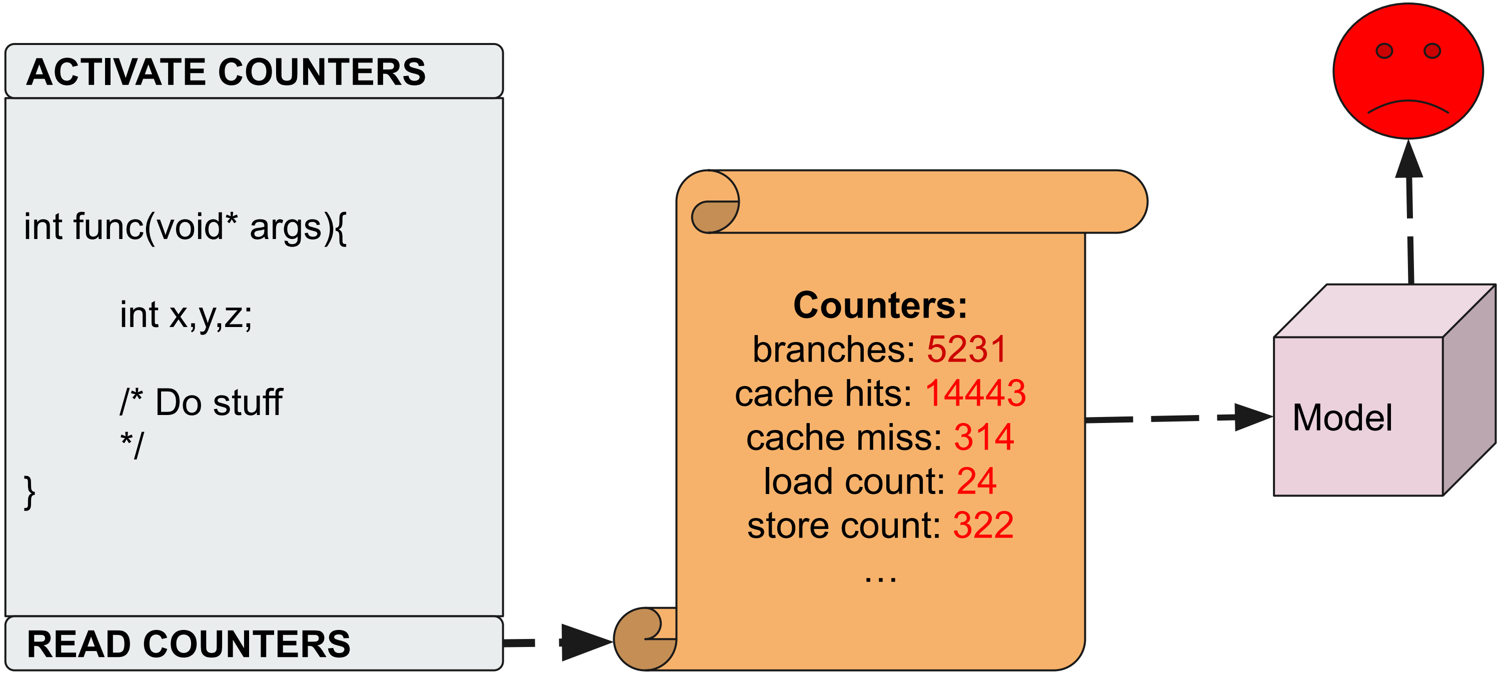
ACTIVATE COUNTERS

```
int func(void* args){
        int x,y,z;
        /* Do stuff
        */
}
```

READ COUNTERS

Counters:
branches: 21
cache hits: 314
cache miss: 3
load count: 24004
store count: 1200
...

Example - Malicious



ACTIVATE COUNTERS

```
int func(void* args){

        int x,y,z;

        /* Do stuff
        */
}
```

READ COUNTERS

Counters:
branches: 5231
cache hits: 14443
cache miss: 314
load count: 24
store count: 322
...

A new approach to interruptible RA

Research proposal

Let’s not interfere with interrupts

Shuffled Measurements and Memory Locking actively stop malware from self-relocating or self-erasing.

Attestation is only aimed at **detecting** anomalies, not **preventing** them.

Idea: do nothing to stop malicious interrupts. Just detect them!

Use the information provided by Performance Counters to detect roving malware.

Performance counters

We will investigate this by relying on existing **performance counters.**

If one has **previous knowledge on interrupts** you can use it to detect malicious activity.

Counters measure the events generated by interrupts during the attestation…

Classification of the results

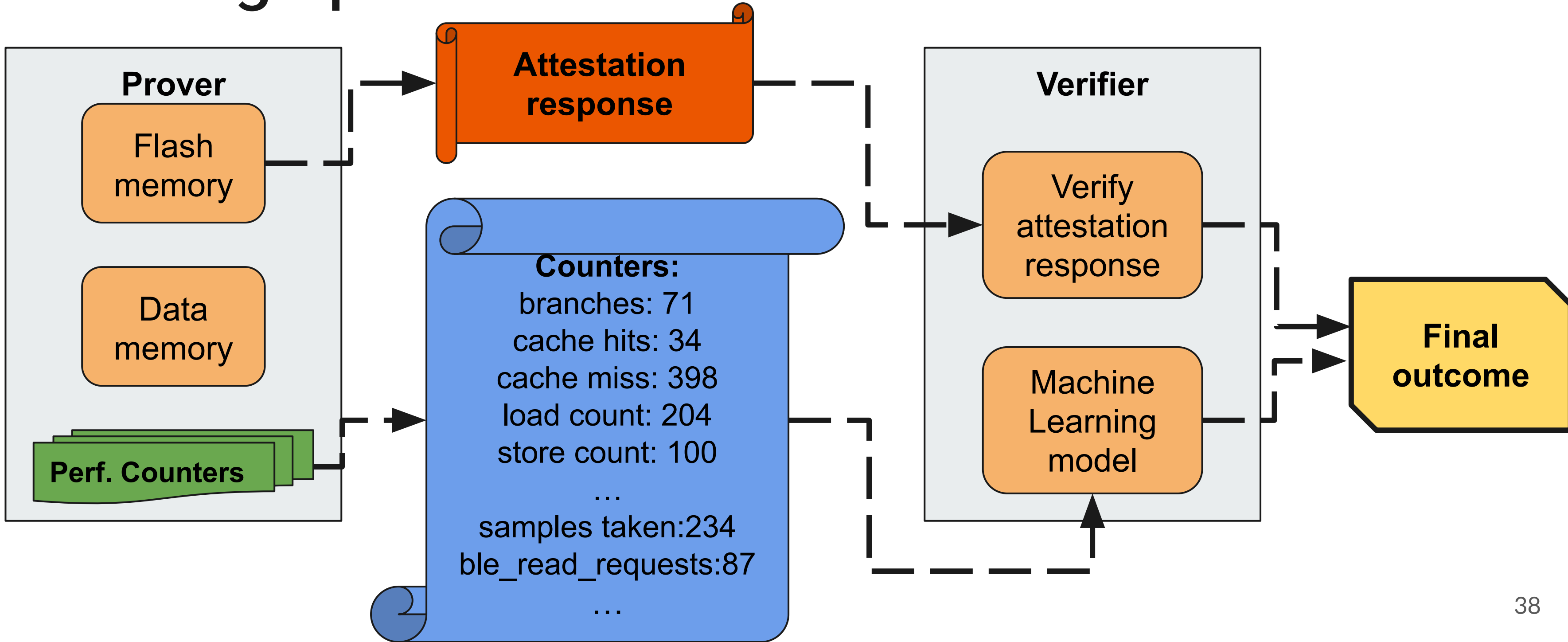
… then feed these values to a suitable a DL/ML model with high accuracy.

We will investigate which model is more suitable for each context.

From **simple models** (Logistic Regression, Decision Trees)...

...to complex ones (Neural Networks, SVM).

Summing-up


Prover
Flash memory
Data memory
Perf. Counters
Attestation response
Counters:
branches: 71
cache hits: 34
cache miss: 398
load count: 204
store count: 100
…
samples taken:234
ble_read_requests:87
…
Verifier
Verify attestation response
Machine Learning model
Final outcome

Innovations in our approach

To the best of our knowledge, nobody has used Performance Counters to detect roving malware during RA.

Attacks only consists of relocating/transient malware, previous works focus on a broad range of attacks.

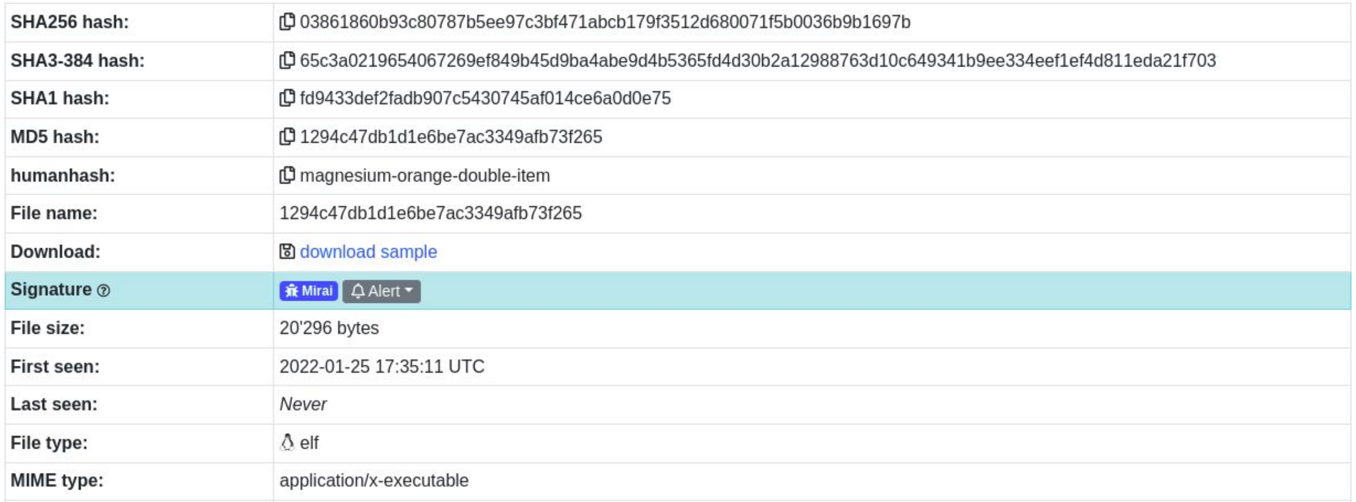
We plan on adding **high-level counters** that monitor application-level events.

Possible because low-end devices only run few simple applications.

Malware for embedded systems

We investigated the features of malware that can run on low-end devices.

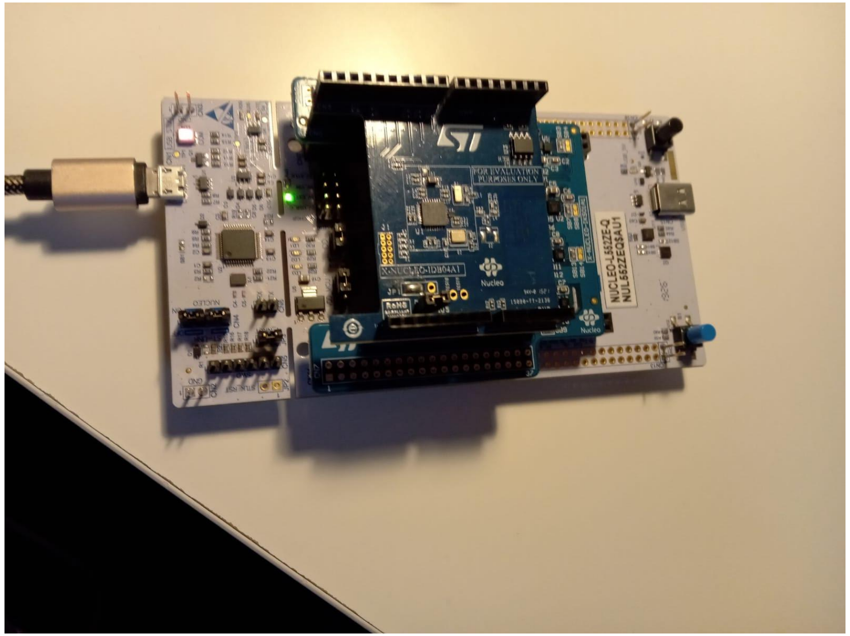
For that purpose, we had a look at some **public malware libraries.**



SHA256 hash:	03861860b93c80787b5ee97c3bf471abcb179f3512d680071f5b0036b9b1697b
SHA3-384 hash:	65c3a0219654067269ef849b45d9ba4abe9d4b5365fd4d30b2a12988763d10c649341b9ee334eef1ef4d811eda21f703
SHA1 hash:	fd9433def2fadb907c5430745af014ce6a0d0e75
MD5 hash:	1294c47db1d1e6be7ac3349afb73f265
humanhash:	magnesium-orange-double-item
File name:	1294c47db1d1e6be7ac3349afb73f265
Download:	download sample
Signature	Mirai Alert
File size:	20'296 bytes
First seen:	2022-01-25 17:35:11 UTC
Last seen:	*Never*
File type:	elf
MIME type:	application/x-executable

Screenshot from MalwareBazaar

Our evaluation node



This node periodically samples Temperature and Humidity values, and broadcasts them using BLE.

We compared 100 sample-and-broadcast iterations with a single malware relocation.

Preliminary outcomes

Evaluation on a Cortex M33 processor, using the 6 “vanilla” Performance Counters in the DWT.

Compared self-relocating malware with an example legitimate activity.

Significant gap in the values measured by the Performance Counters.

The most outstanding one was in the Load/Store clock cycles, as we expected.

What’s next?

- Improve our Proof of Concept implementation (more complexity, variable conditions).
- More use-cases and possibly other evaluation nodes.
- Collect as much data as possible to train ML models
- Generalize the concept to a theoretical methodology, extend to a broader range of applicative scenarios.
- Conclusions: finalize the method, validate it, investigate its limits.

Thanks for your attention!