arm

The challenge of scaling IoT

Modern security for microcontrollers

Gaining user-trust & keeping it

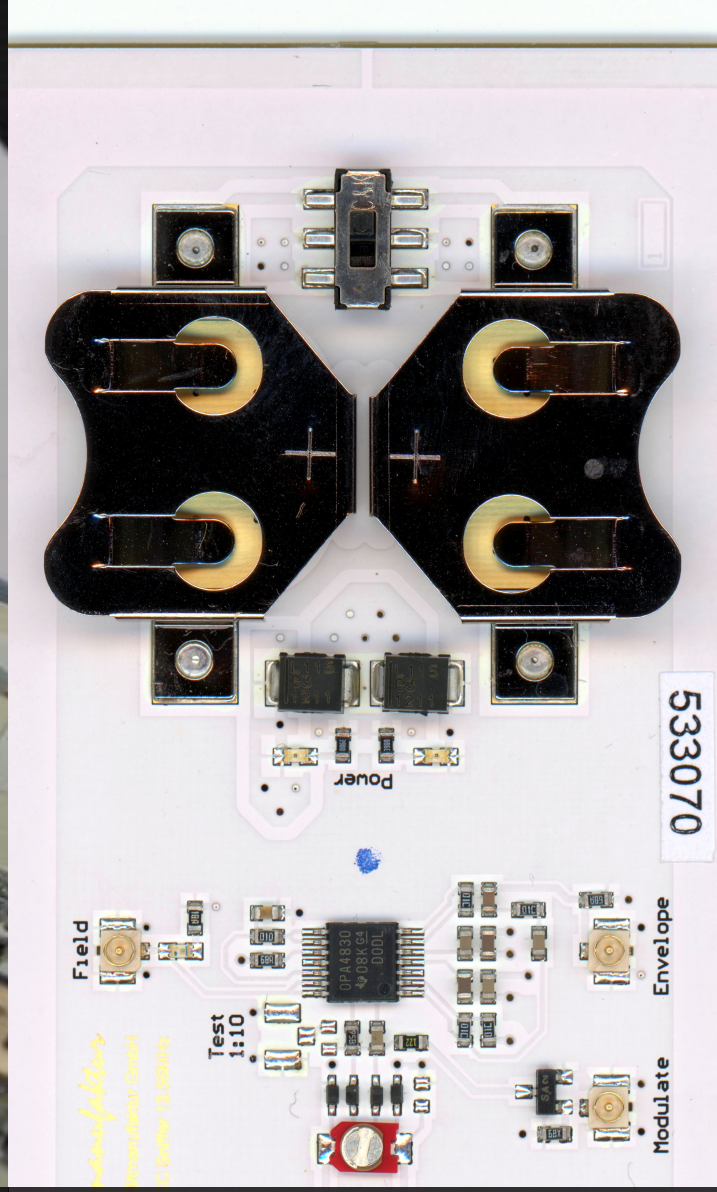
Milosch Meriac
Principal Security Research Lead
milosch.meriac@arm.com

About me & my projects

- Principal Security Research Lead at Arm in Cambridge, UK. Specialized on embedded hardware security.
- Developed the ArmMBED uVisor, a secure hypervisor for Arm Cortex-M microcontrollers.
- Before joining Arm, I've created a range of open hardware designs and software tools around RF(ID) / Bluetooth Low Energy security research and electronic art projects.
- More information on my private projects at meriac.com, Twitter @FoolsDelight & GitHub @meriac.

OpenPCD.org

Open 13.56 MHz RFID NFC Reader



OpenPICC.org

Open 13.56 MHz RFID Sniffer Design with HID iClass decoder

OpenBeacon & SocioPatterns

Realtime 2.4GHz Localization, Human Interaction Analysis & Infectious Diseases Research

Blinkenlights Stereoscope

960 x Realtime 2.4GHz Wireless Halogen Dimmers for Toronto City Hall

Xbox Linux Core Team

Breaking the first serious computing platform for consumers to run Linux

Quick introduction:
Security research at Arm

arm

Collaboration Opportunities with Arm Research

Security-topics we're working on

- Enforcement of device policies with hardware security features, secure protocols and instrumentation of devices and networks
- IoT malware detection (device and network), recovery from malware and containment of affected devices: from very constrained devices, through networks up to data centers
- Enabling end-to-end security/privacy architecture paradigms in cloud service products
- Enabling and promoting automated verification of Arm ecosystem products
- Expanding our tools for automated hypervisor and system verification

arm

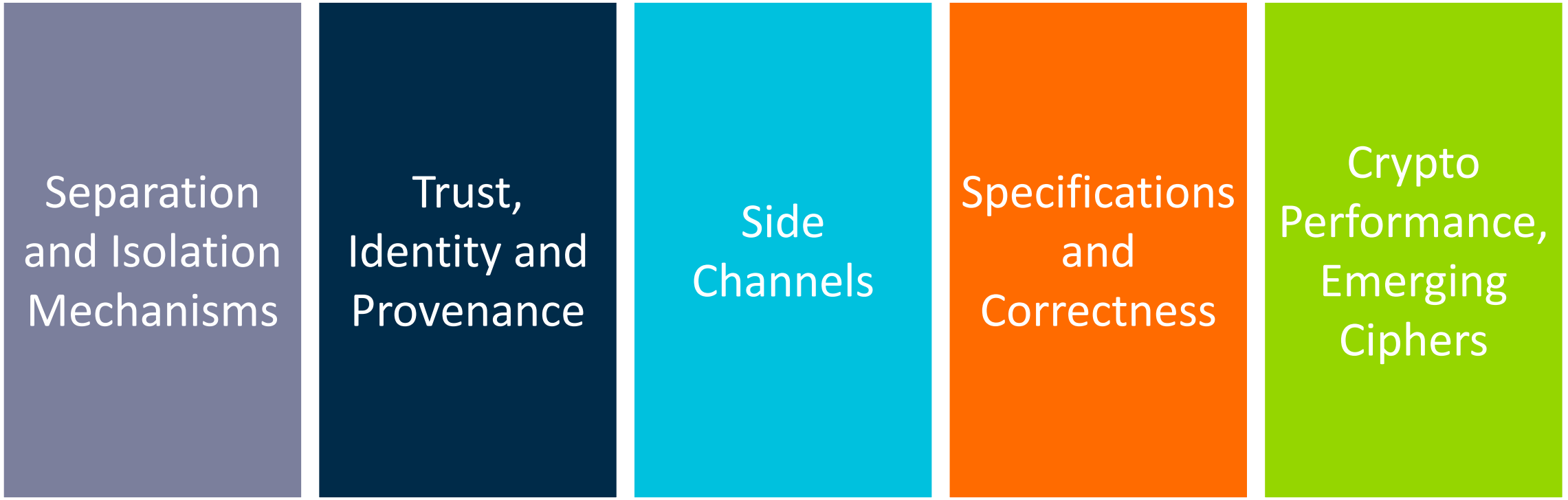
Collaboration Opportunities with Arm Research

Strategic Security Projects

- Support next generation system security architectures in heterogeneous systems
  - E.g. CHERI, seL4 and MirageOS
- Enabling memory authentication and encryption for mutually distrustful edge- & cloud-computing
- Arm architecture side channel evaluations and mitigations
  - Particularly around branching, speculative execution, pipelines and caches.
- Support creation of formally verified secure and safe systems software
  - Improving our verification tools and formal specifications.

Security Research at Arm

Summary



arm

Why is microcontroller security hard ?

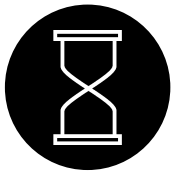
arm

Security
+ time
= comedy

If a product is successful, it will be hacked

System designers must have clear threat models and act accordingly. Threat models must be designed into the system starting at silicon level.

System designers and builders must find all flaws - attackers only have to find one. Old stuff has old bugs – we won't run out of entertainment anytime soon.



Device lifetime

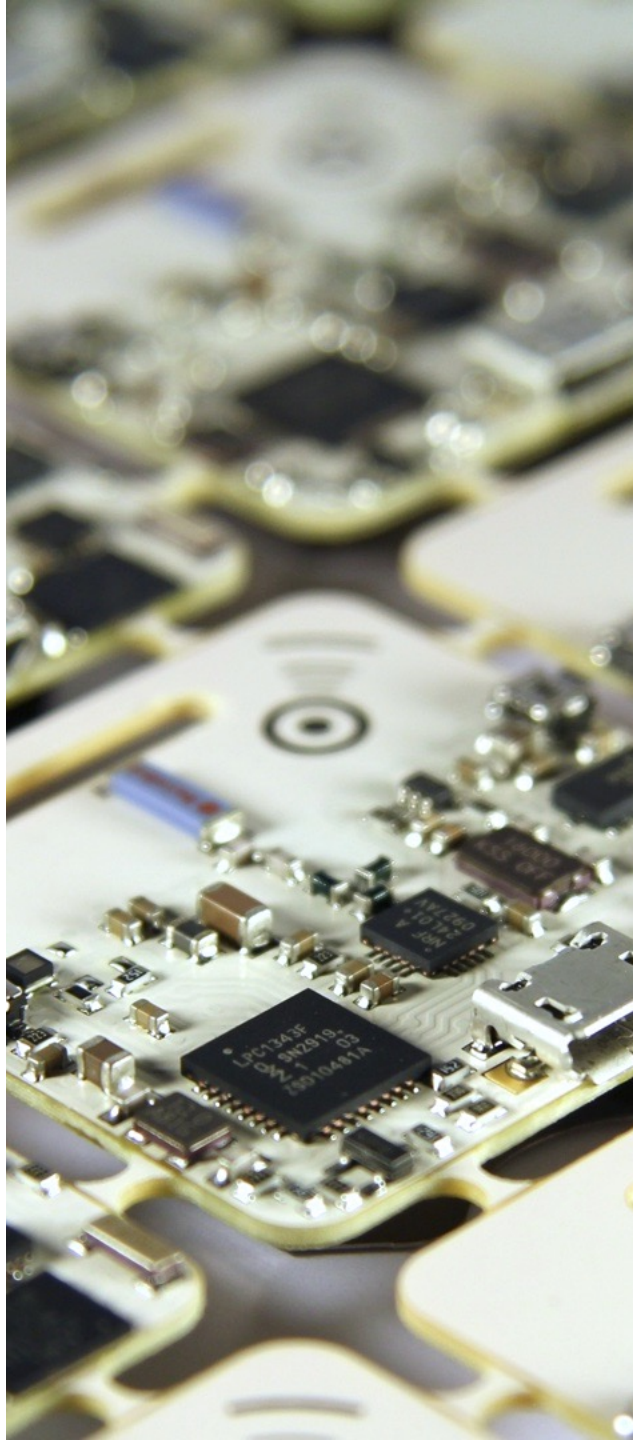
The security of a system is dynamic over its lifetime. Lifetimes of home automation nodes can be 10+ years.

Attacks scale well

The assumption of being hacked at some point requires a solid mitigation strategy for regaining control and simple, reliable and inexpensive updates. Do our defenses scale with our attackers?

Assume you can't prevent being hacked

Value of bugs is expressed by value = number_of_installations x device_value. Increased value and large deployments drive attackers - especially in the IoT. Massively parallelized security researchers & attackers vs. limited product development budget & time frame.

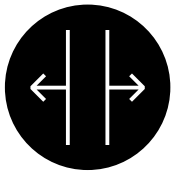


Flat memory models

The 80’s called, and they want their security model back

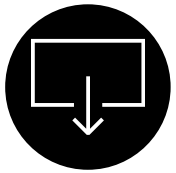
Flat address spaces are common for microcontrollers - resulting from their lack of memory management units (MMUs): yet that does not justify the absence of privilege separation.

Many existing microcontrollers like Arm Cortex-M3/M4 have a hardware memory protection unit (MPU) to compensate for security impact of missing MMU.



No separation

Flat memory models and avoidance of MPUs ignores vital security models like “least privilege”.



Escalation

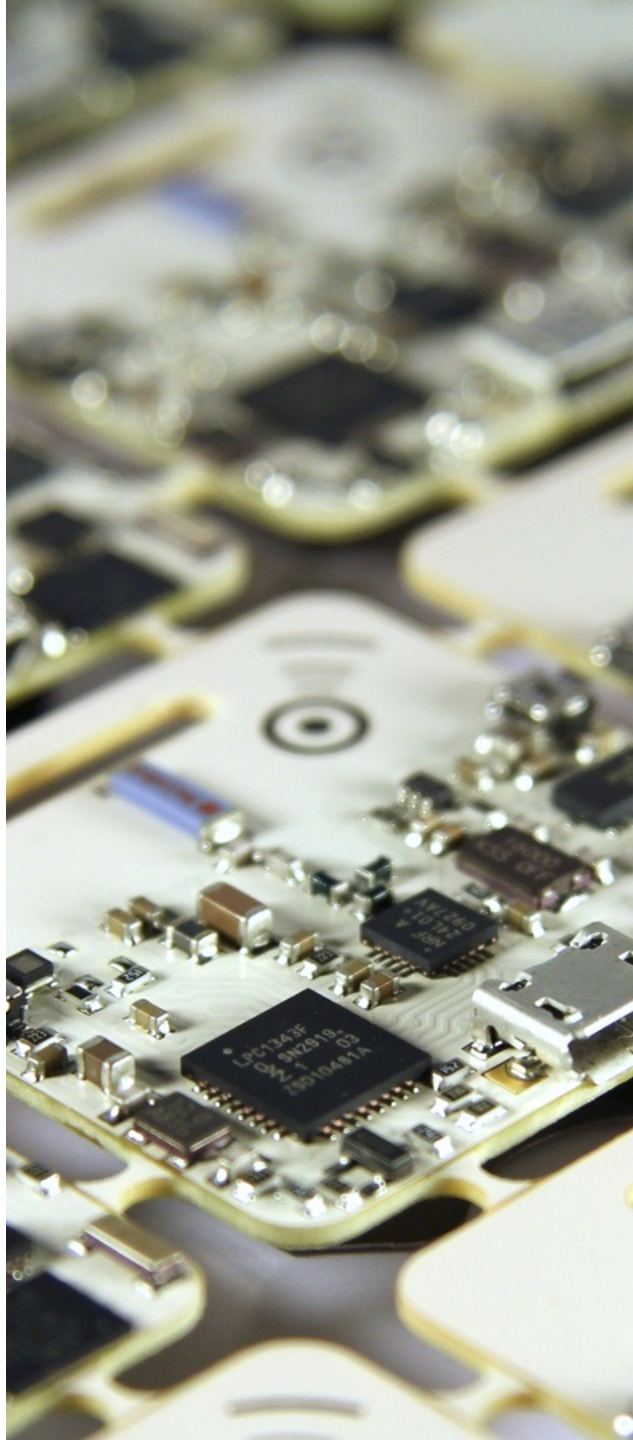
Flat memory models enable escalation & persistence of exploits: Device safety suffers. Uncontrolled writing to flash memories allows persistent exploits across reboots.

Verification

Security verification impossible due to the immense attack surface and lack of secure interfaces between system components.

Leakage

“All your bases are belong to us” – thanks to leakage of identity keys or API tokens attacks are able to progress up the cloud stack.



Data security, seriously?

Check your security assumptions

Microcontroller storage security is still highly diverse and often originates from code-read protection. Yet your data must stay secret, Keys and firmware updates must be secured against tampering and rollback.

Malware must not be allowed to become persistent by installing itself into flash. Sending people out to reset or even re-flash pwnd IoT devices is not an option. We must ensure for device recovery to scale better than the device attacker.

Out of memory

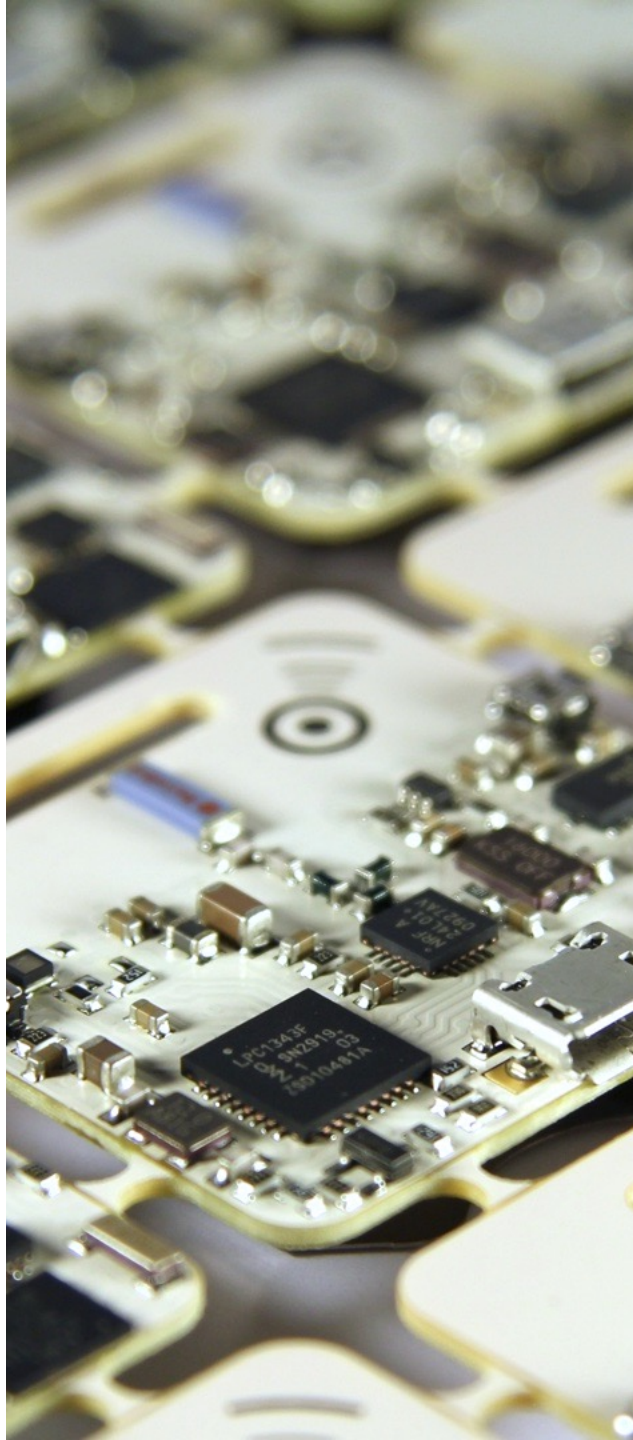
As functionality and wireless stacks grow, microcontrollers run out code memory – firmware downloads must therefore often be buffered externally to prevent device bricking. Update mechanisms are often vulnerable in presence of local attackers controlling the power supply, glitching CPUs or manipulating externally stored data.

Data security?

How confident are you, that your SRAM is erased when resetting the copy protection fuses on your microcontrollers? Additionally data can often be extracted indirectly by using CPU core registers, bootloader bugs or by stepping through code.

Side channels leakage

Microcontrollers often vulnerable to key/data extraction via side channels like power or execution time.



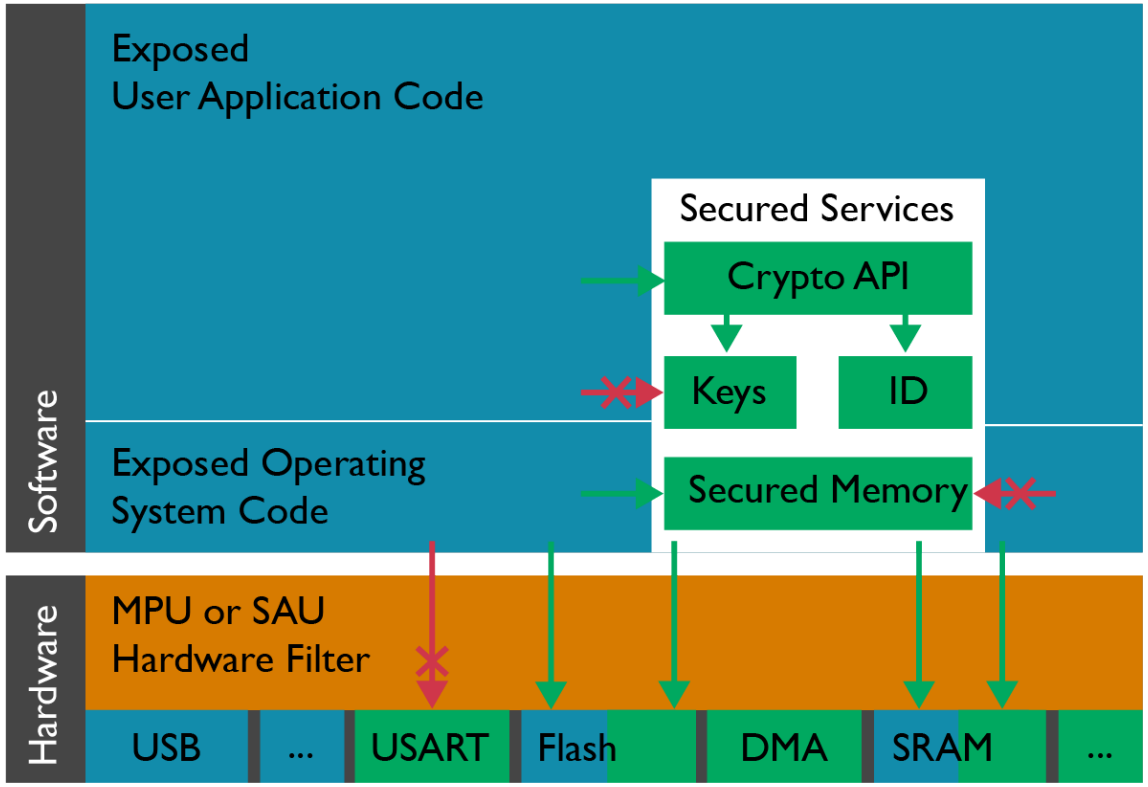
A consistent security model: Privilege separation for low-end Microcontrollers

arm

Security for Microcontrollers?

Consistent security models despite differences in architecture

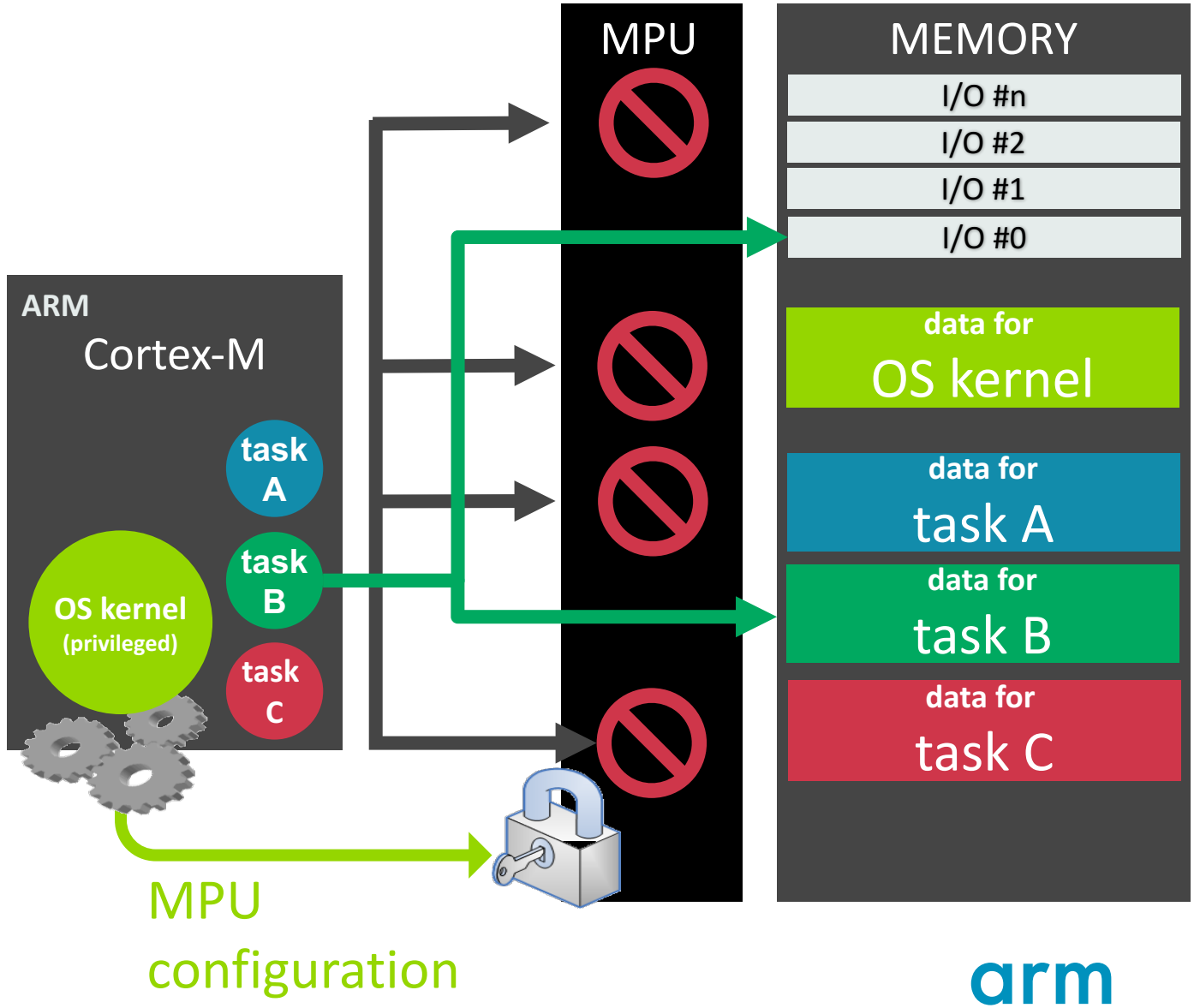
- Compartmentalization of threads and processes on microcontrollers consistent with MMU-enforced application CPU security: Private stack and data sections
- Initialization of memory protection unit based (MPU) on process permissions:
  - Whitelist approach – only required peripherals should be accessible to each box (“Least Privilege”)
  - Each box must have private .bss data and stack sections
- Switch execution to non-secure side/app, continue boot unprivileged to initialize OS and libraries to reduce the attack surface



arm

Memory Protection Units on Arm Cortex-M Microcontrollers

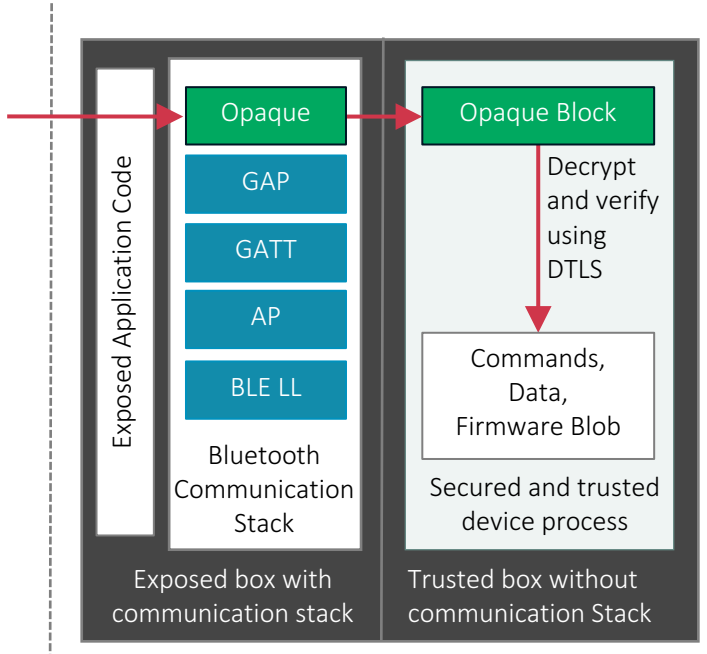
- Can prevent application tasks from corrupting OS or other task data: Improved system reliability through hardware security.
- Enable configurable memory regions
  - Address
  - Size
  - Memory attributes
  - Access permissions
- Optional in all Cortex-M and available on most (except Cortex-M0)



arm

Use case: Secure outbound communication

- Reduce attack-surface of function-critical code and increase bug-resilience
- Trusted messages contain commands, data or firmware parts.
- Box security not affected by communication stack exploits, bugs or infections outside of trusted box.
- Payload delivery is agnostic of protocol stack.
- Resilient box communication over the available channels, protocol-independent:
  - Ethernet, CAN-Bus, USB, Serial
  - Bluetooth, Wi-Fi, ZigBee, 6LoWPAN

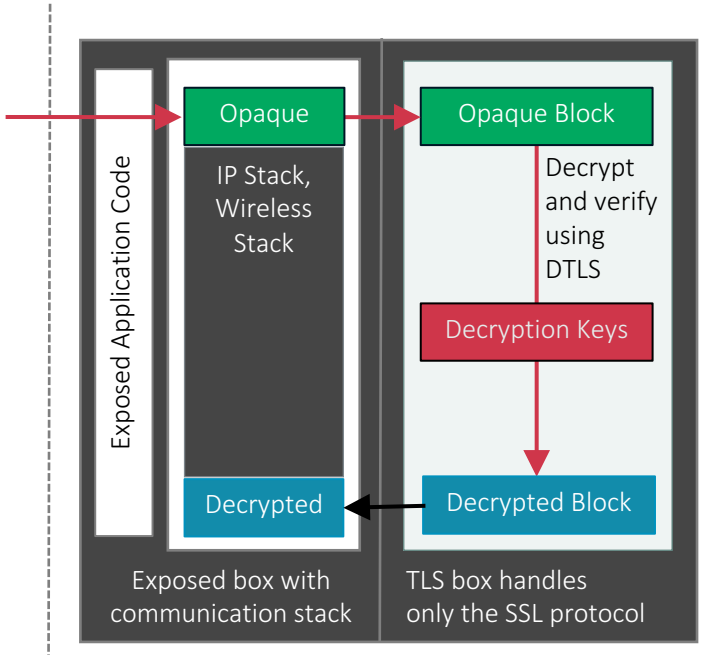


IoT Device owned by user.
Initial identity provisioned by System Integrator
Messages delivered agnostic of communication stack

arm

Use case: Secure server communication & Device identity

- Communication protected using TLS.
- Raw message payloads decrypted and verified directly by protected code:
  - TLS protocol box not exposed to communication protocol stack bugs.
  - No interference by other boxes.
- Client-authentication and encryption keys are protected against malware, key-extraction and identity theft.
- Device-malware cannot interfere without knowing encryption- or signing-keys.

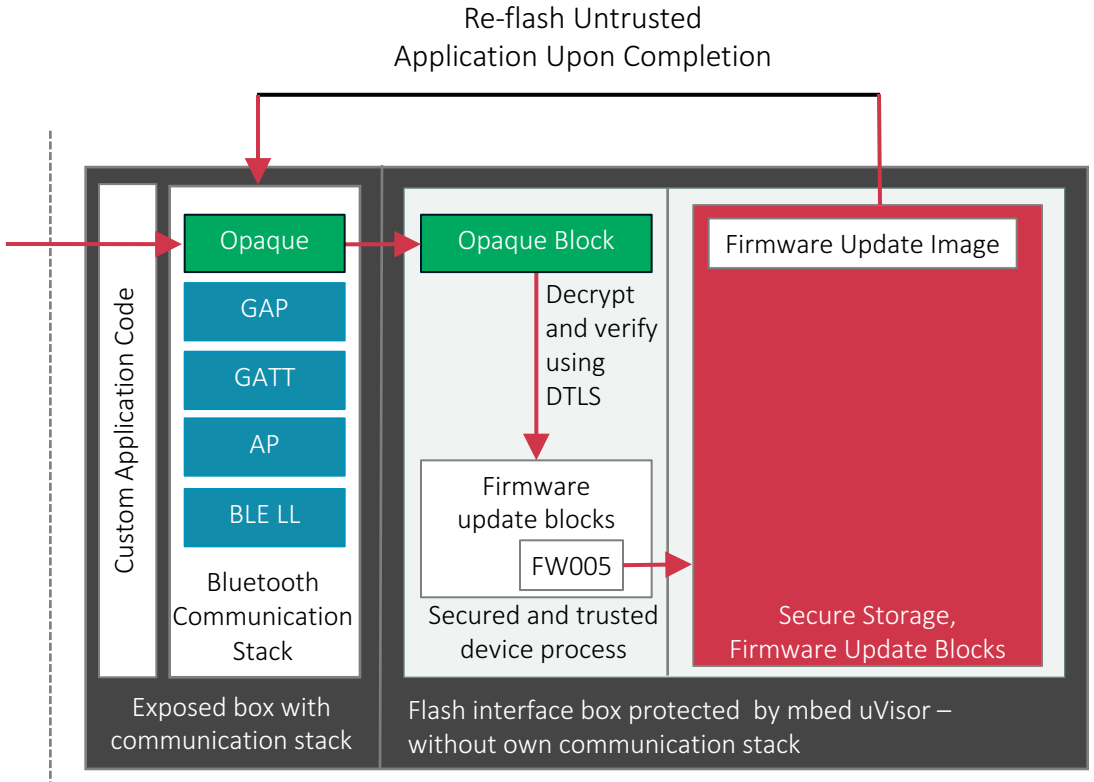


Initial keys provisioned by System Integrator. Messages decoded independent of stacks using mbed TLS in separate security context

arm

Use case: Secure remote firmware update

- Delivery of firmware update must be decoupled from a protocol-independent firmware image verification
- Bugs in communication stacks or cloud infrastructure must not compromise the firmware update:
  - End-to-end security for firmware updates between the firmware developer and the secure box
  - Secure box on the device has exclusive flash-write-access
  - Box with flash controller ACLs only needs the public update key to verify validity of firmware.
  - Local malware can't forge a valid firmware signature to the firmware update box, the required private firmware signature key is not in the device.



IoT device owned by user,
Initial identity provisioned by System Integrator,
Messages delivered independent of stacks

arm

Use case: Controlled malware recovery

Devices can be remotely recovered from malware:

- Enforced communication through the exposed side to the server.
- Thanks to flash controller ACL restrictions, malware cannot modify monitor code or install itself into non-volatile memories.
- Receive latest security rules and virus behaviour fingerprints for detection.
- Share detected pattern fingerprint matches with control server
- Distributed detection of viruses and live infrastructure attacks.

https://commons.wikimedia.org/wiki/File:Biohazard.svg

When communication with server breaks for minimum time:

- Disambiguation from network failure:
  Parts of the device stack are reset to a known-good state.
- Device-reboot prevents malware from staying persistent on the device.
- Device switches to a safe mode after boot to rule out network problems or to remotely update the firmware if needed – and continues operation.

Outlook: Arm TrustZone Technology for Microcontrollers

Bringing Arm security extensions to the embedded world

Optional security extension for the Armv8-M architecture

- **Dramatically simplified security model for Safety-critical device code**
- New security architecture for deeply embedded Cortex-M processors
- Better containerisation of software: Simplified security assessment of embedded devices

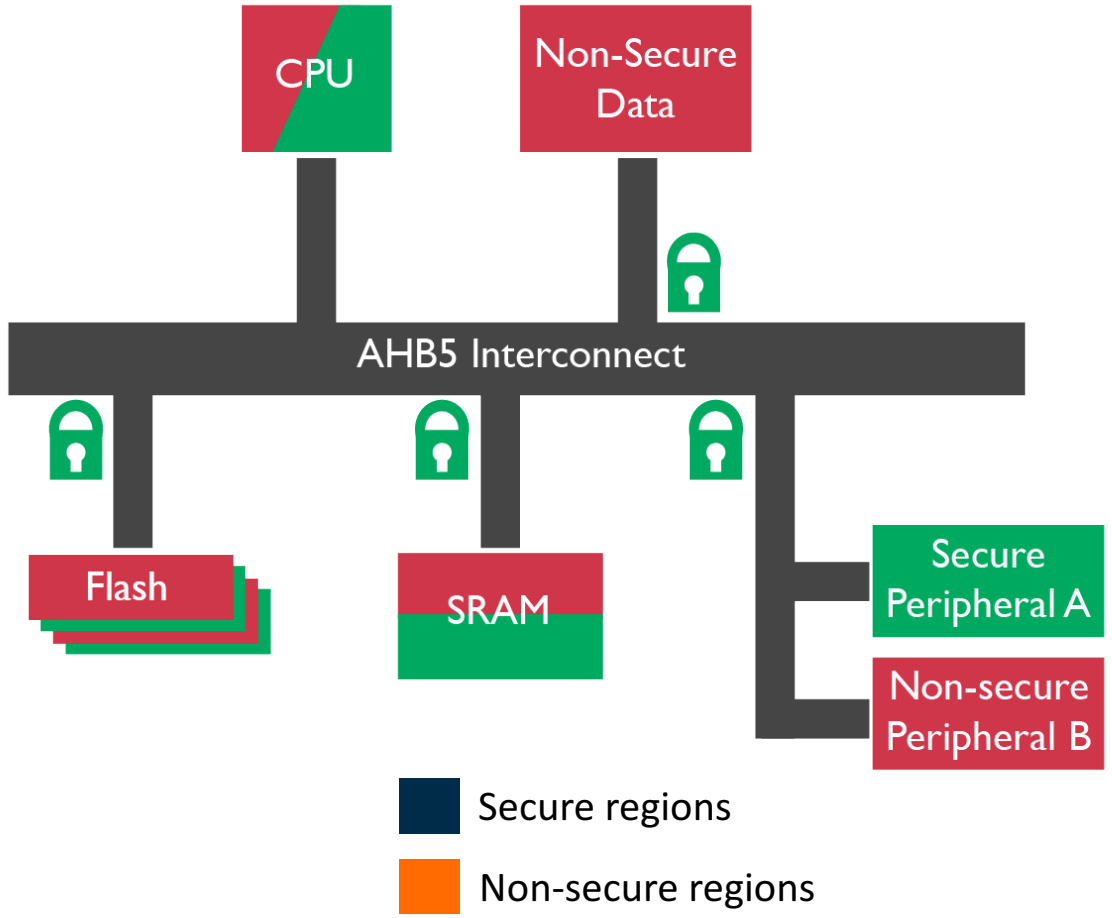
Similar and compatible to existing TrustZone technology

- New architecture tailored for embedded devices
- Adds new security domains with visibility on system/bus level – enabling security-aware peripherals, untrusted drivers and secure DMA.
- Preserves low interrupt latencies of Arm Cortex-M
- Provides high performance cross-domain calling

arm

Applying Arm TrustZone to Cortex-M Microcontrollers

- Armv8-M architecture introduced with upcoming Cortex-M23 and Cortex-M33 microcontrollers
- Extends hardware security features to system level
- Security-aware DMA peripherals and debugging
- Security extended to memories and peripherals through bus filters
  - Memory protection controllers (MPC)
  - Peripheral protection controllers (PPC)
- CPU can run in secure and in non-secure states, with visibility for all bus peripherals
  - Efficient and privacy-enabled transitions between the two security modes
  - Two instances of the interrupt vector table, one for exclusive use on the secure side.
- Stack overflow protection



arm

Scaling Security: Moving towards an IoT Immune System

arm

Moving towards IoT Immune Systems

Overview

- **Detection**
  - Node Instrumentation & Behaviour Sensors
  - Fingerprinting Node Behaviour and System Network Traffic
- **Management**
  - Big data analysis of system behaviour over all customers of a “Security as a Service” provider to weed out false positives and to establish a baseline for device behaviour
  - Live & Continuous “System Health Check”
  - Isolation, Containment and re-flashing of infected Nodes: “IoT Immune System”
- **Prevention, Removal & Containment**
  - Immunization of System – identify Loopholes by using statistical analysis (Device Types, Firmware versions, System Events, Program Execution Patterns, Traffic Patterns) of events leading up to an infection.
  - Block know-malicious signatures on network boundaries using centralized block lists to protect outdated devices.

IoT Immune System Instrumentation

Requirements for IoT Node Malware Detection & Removal

Node Instrumentation

- **CPU level sensors:**
  - Program Counter & Performance counters statistics
  - Detection of Code/Data access patterns
  - Debug-Interfaces (CoreSight TMC etc)
- **Standardized Board level sensors & protocols**
  - CPU/Board Power consumption patterns
  - Inbound/Outbound traffic classification/density
  - Intrusion sensors (Housing, Light Sensors, Chip level etc.)
  - Hardware revision, Product ID, Manufacturer and Firmware version of the device visible to log server
  - Acceleration sensors to detect movement/vibrations resulting from physical attacks

Implementation Details

- Die-Sensors or external watchdogs measure system behaviour
- Sign internal/external measurements plus timestamps and store in cryptographic ledger – can be stored in local network or in offline storage if needed.
- Establish distributed Traffic Sensors for measuring Traffic density and direction – to enable protection within the (mesh) network.
- Signed/Encrypted packets with traffic/behaviour observations are sent back to centralized log servers (local and remote)
- Feed signed/encrypted behaviour measurements back to Application CPU which is responsible for sending the measurements back to centralized servers.
- Non-Compliant applications are reset/re-flashed if they can't prove message delivery cryptographically using crypto-watchdog methodologies.

arm

IoT Immune System Instrumentation

Network Level Immune Reactions & Recovery

Node requirements

- Ability to isolate/quarantine nodes by changing/revoking access keys and tunnelling traffic to a control host (log server?)
- In cases where node functionality is critical, we must enable infected node operation – but maintain network QoS and protect internal and external hosts.
- Ability to reset & re-flash nodes wirelessly – even assuming a malicious application on the Node.
- Standardize FOTA file formats and network protocols
- Prevent firmware downgrade attacks of nodes, with operational exceptions reverting to previously used firmware version over a limited time.
- “Security as a Service” provider can only push node-vendor-signed Firmware – “Least Privilege”
- Establish Watchdog-Style FOTA to externally enforce firmware update of infected systems remotely.

Implementation Details

- FOTA requirements including policy enforcement and downgrade protection are already addressed in the latest Firmware Manifest Format. We started an IETF standardization process already.
- Isolation of nodes can be achieved by implementing software defined networking protocols.
  - In the most simple case networks can be partitioned via Virtual LAN ID’s (VLAN ID) – which are commonly supported by managed routers. It might make sense to extend the concept of Layer VLAN’s to mesh networks. VLAN ID’s can be used End-to-End between Mesh Router and cloud architecture by using L2TP.
  - In the advanced case the (wireless) network applies multiple encryption keys to segment the network and to enforce distinct policies.
- Secure Watchdog for remote recovery from malware infection.

arm

IoT Health Services

Health as a service for the internet of things

Data Processing

- Receive lossy aggregated data from Node and Network sensors.
- To protect the users privacy, pre-aggregation/anonymization of traffic-activity patterns should be done in the local network (edge router etc.)
- Correlate sensor Data across similar devices and similar Firmware revisions
- Detect unusual patterns in data flows (for example an IoT sensor actively scanning the extended network for vulnerable devices) – especially in wireless mesh networks
- Big data analysis across all service customers to weed out false positives. Behavioural changes due to different firmware revisions can be detected more reliably
- Easier to detect unusual patterns in larger data sets due to more data on devices with similar configuration

Actions taken

- Trigger Network immune reactions depending on calculated likelihood
- Optionally allow the Health Service to take action on your behalf (trigger firmware update, isolate nodes etc.).
- Optionally disambiguate security warnings (Incidents of lower likelihood) by human operators for premium customers – and take extended action if necessary.
- After identifying Infection Vectors, distribute traffic/behaviour signatures for devices without patches
- Enforce policies so devices with unpatched devices can only be reached by filtered data (keeping them functional, but safe at higher latencies – Remote Man-In-The middle firewall / DPI).
- Run filters at trusted border router to protect customers privacy. After an infection is detected, customer gets choice to involve cloud service into recovery efforts.

arm

Trusting IoT Health Services

Edge Computing Appliances for Resilience and User Privacy Protection

- **Objective**
  - Establish trusted cloud computing appliances that will be deployed to the local network for greater availability and trust. Appliances can be as simple as hardened routers running simple software process or high-end tamper-proof 19” rack servers.
  - Establish Trusted Execution Environment (TEE) boxes that can be remotely provisioned (cloud or owner) – hardened against local physical attacks (cold boot attacks, malicious board level modifications & attacks)
  - Cloud services can push down low-latency jobs, computation tasks, machine learning, application of machine learned models or for mirroring data speculatively from or to the local network.
  - Platform guarantees integrity and security across multiple applications/VM’s running on the device and protects against users tampering with the device locally (exfiltration of data, changing behaviour etc.)
  - Increase the resilience of cloud applications: Core functionality of a cloud application can work without network connectivity. See Amazon AWS Greengrass as best-practice example for accessible compute at the edge.
  - Support low power devices with trusted computing resources as important tasks can be migrated to AC-powered appliance in the local network.
- **Hardware Security Requirements**
  - Encrypt and authenticate external memory interfaces (DDRAM, Flash) for operation in mutually distrustful environments
  - Establish hardware root of trust back to the SoC-manufacturer to prevent emulation-attacks

arm

IoT Immune System & Health Services

Summary

- Reduce Complexity of the local system/network security to **standardized** sensors and Immune Reaction Interfaces.
- Enable Big-Data Security Analysis of Malware detection sensors – reduce noise by comparing similar devices across all customers while preserving the customers privacy. Detect infection vectors statistically by comparing commonalities of different hosts running the same firmware to establish baseline
- Monitor spread of malware and infer infected hosts from historic traffic metadata to coordinate node isolation.
- Aggregated analysis in cloud services allows continuous improvement of security rules. These rules can be exported back to edge computing notes for greater privacy and resilience.
- Rules / filters which are verified in cloud services can be enforced with low overhead at mesh level to prevent local spreading of malware.

arm

Questions & Further Reading

- Please contact me at milosch.meriac@arm.com with your questions and suggestions. **We're hiring – please send me a mail!**
- The **ARM IoT Security Manifesto**, describing an IoT Immune System: https://pages.arm.com/iot-security-manifesto.html (Set of whitepapers)
- The Arm **Platform Security Architecture**:
  https://developer.arm.com/products/architecture/platform-security-architecture
  https://pages.arm.com/PSA-Building-a-secure-network.html (Whitepaper)
- High-End Security for Low-End Microcontrollers:
  Hardware-Security Acceleration on Armv8-M Systems
  https://doi.org/10.5281/zenodo.571159 (Slides)
  https://doi.org/10.5281/zenodo.571158 (Whitepaper)

arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद