Towards Trusted Extensible Device Measurement and Management via Intra-Firmware Privilege Isolation

Chuqi Zhang[§], Bonan Ruan[§], Vikram Ramaswamy[¶], Zhenkai Liang[§], Adil Ahmad[¶]
National University of Singapore[§], Arizona State University[¶]

1 Background

The rapid expansion of interconnected smart devices has spurred diverse applications—from consumer technologies like smart city infrastructures to healthcare, wearables, and home automation. Advancements in industrial automation, robotics, augmented reality, and autonomous vehicles further underscore their growing impact, especially in mission-critical systems such as cyber-physical systems (CPSs). To support these applications, increasingly complex functionalities are integrated onto a single system-on-chip (SoC).

Firmware, the core software embedded in these smart devices, operates at the highest privilege level to support platform-specific tasks. However, its large codebase introduces considerable vulnerabilities. Once the firmware is exploited, the entire device is compromised by attackers. Recent studies have continuously revealed such vulnerabilities [3, 4]. Given these threats, *how can we assess the runtime security state of critical firmware and reliably reset a device when a compromise is detected, according to security policies?*

2 System and Threat Model

We consider two components during normal operations: *the device and the remote verifier*. The remote verifier is a trusted entity operating on backend servers or in the cloud, responsible for managing interconnected devices [7]. We adopt the following assumptions and consider these attack vectors:

- Untrusted device firmware. While initially benign, the firmware is vulnerable due to bugs, making it susceptible to full compromise by external remote attackers. Similarly, while cryptographic protocols and key management schemes are secure, they remain susceptible to firmware compromise. If compromised, an attacker can abuse cryptographic keys to impersonate the device and tamper with its communication channel with the remote verifier.
- Trusted early boot process. The early boot process is always trusted, as the firmware bootloader images are permanently burnt and stored in the secure regions (trusted boot ROM and SRAM) [7, 10].

Out-of-scope. We assume the hardware is trusted. Physical attacks and micro-architectural side-channels are out of scope.

3 Goals and Requirements

Our goal is to *design an extensible device measurement framework, allowing remote verifiers to register multiple services and reliably inspect a device's runtime security states and recover it, regardless of device firmware compromise.* To achieve this, the following requirements must be met:

R1. Isolated on-device monitor. A secure, most privileged monitor must reside on the device. The privilege allows it to capture firmware-level states (e.g., runtime memory, CPU, and register contexts). Its isolation ensures state integrity even if the firmware is compromised.

R2. Extensible firmware state monitoring. The framework should allow the verifier to dynamically adjust monitoring services at runtime, measuring diverse device states in accordance with evolving security policies.

R3. Secure communication channel. Communication between the device monitor and the remote verifier must be authenticated and immune to impersonation. The verifier must attest to the monitor to ensure digest integrity, while the monitor must verify remote requests to protect privacy.

R4. Non-deniable device recovery. Upon receiving state digests from the monitor, the verifier must assess the device's security and, if compromised, force a reset to a benign initial state within a controllable, non-deniable delay.

4 Intra-Firmware Isolation

At the core of the privileged on-device monitor is an intra-firmware isolation design. This design partitions the firmware privilege level into two compartments: the deprivileged *Firmware* and the high-privilege *Monitor* [2]. Fundamentally, isolation is achieved using hardware memory protection primitives—such as ARM's Permission Indirection Extension (PIE) [1] and RISC-V's enhanced PMP (ePMP)—which restrict the CPU's accessible memory regions.

Hardware-assisted memory isolation maintains a protected region for the Monitor's code and data, enabling a paravirtualization-like mechanism that intercepts and handles predefined privileged instructions [2, 6, 9]. This mechanism enforces the following security invariants through software instrumentation, traps, and binary verification:

I1. Firmware and Monitor are partitioned into two distinct physical regions. The code and data sections of Firmware and Monitor are distinct at compilation. During early boot, the Monitor memory is protected (by configuring the hardware memory protection primitives). Once the Monitor is loaded, it also enforces the memory protection primitives to always restrict Firmware's access permissions (*I2*).

I2. All privileged instructions and control interfaces are only inside Monitor. At compilation, the Firmware is instrumented,

replacing privileged instructions with *secure calls* to the Monitor. Such control interfaces/instructions include system registers (e.g., used to configure memory isolation primitives), exception vectors (to control the interrupt flows), and certain MMIO regions (to control devices, explained in §7).

I3. Firmware can only call the Monitor through fixed secure call gates. The gates serve as explicit privilege transition boundaries. At such gates, the Monitor securely saves and restores current contexts, verifies the requested privileged operations, and executes them on behalf of the Firmware. Note that such gates are deterministic and atomic, ensuring the privilege is only preserved during executing the Monitor, and cannot be redirected to the Firmware even at interrupts [2].

5 Extensible Firmware Monitoring

The trusted Monitor, residing at the firmware hardware privilege level (§4), is able to monitor and profile the Firmware, and the whole-system states, at runtime.

An eBPF-like mechanism is integrated into the Monitor to support extensible and dynamic hot-pluggable monitoring. In particular, the remote verifier specifies the target device states to be traced, and programs the *tracer-programs*. Such tracer-programs are sent to the Monitor, which loads them to collect runtime device behaviors. We illustrate typical tracer-programs, which could be applied according to the security policies, in the following paragraphs.

- Memory scanner. This scans firmware memory (e.g., executable sections) to detect modifications. For simple devices, any alteration to the firmware code is identified and reported to the remote verifier.
- Firmware control flow attestation. This allows fine-grained tracking of runtime firmware execution, by accumulating control flow information into a hash measurement. The firmware can be instrumented (at selected control flow graph edges [8]) to call the Monitor. Alternatively, hardware tracing (e.g., ARM Embedded Trace Macrocells) can be employed using a protected control interface (*I2*, §4).
- Full system auditing. This allows system-wide behaviors (beyond the low-level firmware activities) to be captured. For instance, the Monitor may intercept and trace runtime device network event packets [10].

6 End-to-End Secure Communication

All communications between the Monitor and the verifier are authenticated and transmitted over a secure channel.

A two-way secure channel is established by generating and exchanging cryptographic key pairs during machine provisioning [7]. Specifically, the verifier's public key is stored in the Monitor's data region, enabling the device to authenticate signatures. Meanwhile, the device's key pair—derived from the Device Identifier Composition Engine (DICE) [7]—allows the verifier to retrieve the device's public key at provisioning.

During firmware boot, the trusted early boot process and the Monitor are loaded first, generating a sealed firmware hash using the cryptographic key before any untrusted firmware code is executed. This enables the verifier to perform remote attestation before establishing the channel.

7 Secure Timer-Enforced Device Recovery

At device initialization, the Monitor is configured with a protected watchdog timer, with a verifier-controlled interface, to enforce a secure, controllable device reset.

Towards a protected watchdog timer, the Monitor traps the timer and locks it down (*I2*, §4), by protecting its control interfaces (e.g., MMIO regions) and exclusively controlling its interrupt handler [5, 10]. By doing so, the remote verifier is able to configure a *hard device reset deadline*, such as a one-hour interval, which periodically forces the device to be recovered to its initial state, regardless of whether the firmware is compromised or not.

To avoid frequent and unnecessary device resets (when it is benign), only the verifier is allowed to communicate to the Monitor, which feeds the watchdog and extends the reset deadline (through the secure channel, §6). In particular, the verifier periodically receives and evaluates device state digests (§5). Upon successful measurement, the verifier would extend the reset deadline to keep the device alive [7].

References

[1] Permission indirection and permission overlay extensions. https://developer.arm.com/documentation/102376/0200/Permission-indirection-and-permission-overlay-extensions.

[2] Mark Kuhne, Stavros Volos, and Shweta Shinde. Dorami: Privilege separating security monitor on risc-v tees. In *Usenix Security*, 2024.

[3] Christian Lindenmeier, Mathias Payer, and Marcel Busch. EL3XIR: Fuzzing COTS secure monitors. In *USENIX Security*, 2024.

[4] Tobias Scharnowski, Nils Bars, Moritz Schloegel, Eric Gustafson, Marius Muench, Giovanni Vigna, Christopher Kruegel, Thorsten Holz, and Ali Abbasi. Fuzzware: Using precise MMIO modeling for effective firmware fuzzing. In *USENIX Security*, 2022.

[5] Jinwen Wang, Ao Li, Haoran Li, Chenyang Lu, and Ning Zhang. Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone. In *IEEE S&P*, 2022.

[6] Wenhao Wang, Linke Song, Benshan Mei, Shuang Liu, Shijun Zhao, Shoumeng Yan, XiaoFeng Wang, Dan Meng, and Rui Hou. The road to trust: Building enclaves within confidential vms, 2024.

[7] Meng Xu, Manuel Huber, Zhichuang Sun, Paul England, Marcus Peinado, Sangho Lee, Andrey Marochko, Dennis Mattoon, Rob Spiger, and Stefan Thom. Dominance as a new trusted computing primitive for the internet of things. In *IEEE S&P*, 2019.

[8] Nikita Yadav and Vinod Ganapathy. Whole-program control-flow path attestation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[9] Chuqi Zhang, Rahul Priolkar, Yuancheng Jiang, Yuan Xiao, Mona Vij, Zhenkai Liang, and Adil Ahmad. Erebor: A drop-in sandbox solution for private data processing in untrusted confidential virtual machines. In *ACM EuroSys*, 2025.

[10] Chuqi Zhang, Jun Zeng, Yiming Zhang, Adil Ahmad, Fengwei Zhang, Hai Jin, and Zhenkai Liang. The hitchhiker's guide to high-assurance system observability protection with efficient permission switches. In *ACM CCS*, 2024.