Hardware Anomaly Detection in Microcontrollers Through Watchdog-Assisted Property Enforcement

Maksym Melnyk*, Jacob Thomas*, Max Wandera†, Ajesh Koyatan Chathoth†, Michael Zuzak*
*Rochester Institute of Technology, Rochester, NY USA
{mm6878, jbt5011, mjzeec}@rit.edu
†Eaton Corporation, Pittsburgh, PA USA
{MaxBWandera, AjeshKoyatanChathoth}@eaton.com

***Abstract*—The development of anomaly detection and trust mechanisms for low-end microcontroller units (MCUs) has received substantial attention. Prior approaches generally explore either hardware or co-design-based security techniques to secure low-end devices. However, in some cases, the necessary hardware support is more expensive than the MCU itself, rendering these approaches infeasible in some applications. In this work, we propose a novel security mechanism that adapts prior custom-hardware-assisted trust mechanisms to leverage only standard on-chip hardware for anomaly detection and trust in low-end MCUs. Specifically, we propose a runtime security property enforcement mechanism that periodically checks user-defined security properties to detect anomalous behavior using hardware watchdog (HWD) timers. Since HWD timers are standard in most low-end microprocessors, no additional hardware modifications are necessary. For evaluation, we implemented the proposed anomaly detection framework in an ARM Cortex-M4 device. A set of 11 MITRE Common Weakness Enumeration (CWE) benchmarks were implemented and executed on the MCU to evaluate the approach. All benchmarks were detected within 40ms, with a corresponding memory overhead of 5.1kB and performance overhead of less than 0.1%, highlighting the detector's practicality and low overhead.**

***Index Terms*—IoT Security, Hardware Anomaly Detection, Attestation, HWD-Assisted Property Enforcement**

I. Introduction

Embedded devices are increasingly deployed in diverse applications, ranging from IoT devices to industrial automation. These devices often operate on sensitive data and perform safety-critical tasks [1], making them attractive targets for both software [2]–[4] and hardware [5] attacks. Therefore, ensuring the trust and security of these devices is vital.

Due to highly constrained design and cost budgets, low-end MCUs generally lack hardware features found in more resource-rich systems, such as trusted execution environments and process isolation. This limitation has driven the development of various trust and security mechanisms for resource-constrained systems, including remote attestation [6]–[9] and hardware property enforcement [10]–[12]. Prior research has shown that the necessary assumptions for software-based attestation schemes are generally infeasible in practice [13]. Consequently, these approaches generally rely on either hardware-assisted or co-design approaches that incorporate additional custom hardware into the device to perform anomaly detection and ensure trust. These hardware modifications may be impractical for low-end MCUs because: 1) the cost of such hardware-assisted security mechanisms may exceed the cost of the MCU core itself [8], and 2) device application developers may lack the capabilities or design details for re-design.

This motivates our work. We consider the case where a low-end MCU is deployed in a safety-relevant application where trust is important, but the designer or user of the device is unable or unwilling to modify the hardware. In this case, existing trust and anomaly detection mechanisms that require hardware assistance, such as attestation [6]–[9] or hardware property enforcement [10]–[12], cannot be used. To address this, we aim to adapt hardware property enforcement to a less invasive implementation that uses standard on-chip hardware to detect anomalies and facilitate trust in the device.

The role of hardware assistance in prior work is to ensure that the trust mechanism cannot be disabled, even if an adversary has seized control of the MCU. To address this, we propose employing a hardware watchdog (HWD) timer to ensure security properties are periodically checked/enforced. A HWD is an always-on hardware timer that must be periodically reset ("fed") to avoid a mandatory system reset. These timers are standard in most low-end MCUs (e.g., MSP430, STM-series ARM Cortex-Mx). They are enabled during secure boot, and cannot be disabled thereafter [14].

To periodically feed the HWD, a non-maskable interrupt (NMI) will be configured to run at regular intervals. This interrupt will trigger a handler that checks a set of predetermined logical security properties against the processor state. The NMIs will be periodically generated by a hardware-based interrupt request. These security properties are logical formulas which characterize secure MCU states. They are evaluated in the NMI handler and assess the security-relevant state of the MCU (e.g., link register (LR), program counter (PC), configuration registers, etc.) using these formulas. If none of the properties are violated, the interrupt feeds the HWD timer; otherwise, the HWD times-out, initiating a mandatory shutdown sequence and notifying the user of the violated security property. By doing so, the HWD provides hardware assistance to ensure that security properties are periodically enforced without custom hardware redesign.

Contribution: We design, implement, and evaluate HWD-assisted property enforcement, a hardware-assisted trust and anomaly detection framework for low-end MCUs. The proposed approach relies on a HWD timer to periodically enforce

logical security properties. Because HWD timers are standard in most low-end MCUs, the proposed trust mechanism functions without the need for custom hardware re-design, which was required by prior work [6], [7], [9]–[12]. To evaluate HWD-assisted property enforcement, we develop a set of 8 properties to be enforced in low-end MCUs that address 13 common MITRE common weakness enumerations (CWEs) [15]. These properties are implemented via HWD-assisted property enforcement in an STM-series ARM Cortex-M4 (STM32L475) with a 40ms property checking interval. Examples of each MITRE CWE were executed separately on the MCU. All property violations were detected within 40ms, which is determined by the time to the next property check, with a RAM overhead of 5.1kB and a performance overhead under 0.1%.

II. Related Work and Preliminaries

A. Remote Attestation

Remote attestation (RA) is a challenge-response security protocol that allows a trusted verifier to assess the integrity of a prover's memory [9]. Figure 1 provides an overview of RA. The process involves four steps, numbered (1)-(4). First, the verifier sends a challenge containing a secret-derived token and a request for an integrity check. Second, the prover authenticates a predefined region of memory, often by hashing the memory contents and including some aspect of the challenge. Third, the prover returns this response to the verifier. Finally, the verifier checks the response to assess memory integrity.

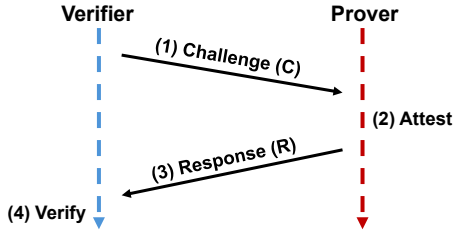

Fig. 1: An overview of remote attestation (RA).

RA-based approaches have shown success in detecting various attacks, including data-oriented [2] and control-oriented [3], [4] attacks . RA solutions can be software-based [16], [17], hardware-based [18], [19], or hybrid [6], [8]. However, hardware-based approaches are often prohibitively costly for low-end MCUs [8]. Conversely, software-only mechanisms require strong assumptions, making them largely infeasible in practice [13]. Several hybrid approaches have succeeded in low-end MCUs, but inherently require hardware support. The proposed HWD-assisted trust mechanism in this work is similar to RA-based approaches, with the HWD essentially serving as an on-chip verifier for the MCU.

B. Hardware Property Enforcement

Another approach to anomaly detection in processors employs dedicated hardware support to enforce a security policy on a system, which we refer to as hardware property enforcement [7], [10], [11]. For example, SPECS [10] and FinalFilter [11] propose modifying the processor pipeline to automatically check a logical security policy and repair any violations at runtime. While such approaches are effective, they rely on hardware modification, which may be infeasible for low-end MCUs. The proposed HWD-assisted trust mechanism in this work relies on a similar formulation of security properties and invariants enforced on the processor for anomaly detection.

C. Threat Model

We consider a scenario where an adversary aims to compromise a simple, low-end MCU (e.g., TI MSP430, ARM Cortex-M0/4). Both software (e.g., malicious code execution) and hardware (e.g., fault injection) attack modalities are within scope. The adversary's objective is to maliciously modify the configuration or state of the victim MCU. More formally, we define attack success as any modification of protected MCU state or memory. The attacker can employ any strategy using:

- Physical access to a functional, deployed version of the MCU. The adversary can interact with the MCU to understand its operation, including input-output behavior.
- Network access to the device, including the ability to read and write packets to the device network.
- The ability to interact with communication interfaces (e.g., JTAG, I2C, etc.). The ability to use these interfaces does not imply that the data will be processed.

A successful defense must detect any in-scope attack and notify the user of a security violation in under 40 ms. We consider the MITRE CWEs [15] in Table I to be the minimal set of in-scope threats. This set contains a set of CWEs that impact MCU memory or configuration registers from the *Top 25 Hardware CWEs* list from MITRE. For evaluation, a successful defense strategy is defined as one that mitigates test instances of each CWE.

TABLE I: List of considered MITRE CWEs.

CWE [15]	CWE Description
CWE-121	Stack-based Buffer Overflow
CWE-284	Improper Access Control
CWE-269	Improper Privilege Management
CWE-506	Embedded Malicious Code
CWE-1191	Improper Access Control for On-Chip Debug
CWE-1231	Improper Prevention of Lock Bit Modification
CWE-1233	Security-Sensitive Hardware Controls with Missing Protection
CWE-1240	Use of a Cryptographic Primitive with a Risky Implementation
CWE-1244	Asset Exposed to Unsafe Debug Access
CWE-1256	Improper Restriction of SW Interfaces to HW
CWE-1260	Improper Handling of Memory Overlap
CWE-1272	Sensitive Information Uncleared Before State Transition
CWE-1274	Improper Access Control for Memory Containing Boot Code

III. Watchdog-Assisted Property Enforcement

While both RA and hardware property enforcement ensure MCU trust and integrity, they require dedicated hardware support and may necessitate redesigning parts of the pipeline. [10], [11]. Re-architecting processor hardware for specific applications is costly and often infeasible, especially for the low-end hardware considered in this work. Thus, we aim to adapt these approaches to develop *a generic security mechanism that provides hardware-assisted enforcement without custom hardware support*.

Specifically, we propose a property enforcement mechanism similar to hardware property enforcement, as outlined in Sec. II-B, but without requiring custom hardware redesign. The main challenge is ensuring the execution of the security mechanism, even if malicious hardware or software takes control of the MCU. To address this, we explore the use of a HWD timer to provide standardized hardware assistance. A HWD timer is an always-on hardware timer that must be periodically fed to avoid a mandatory reset. These timers are standard in low-end MCUs and often cannot be disabled after activation [14]. This allows periodic security property checks to be linked to feeding the HWD, thereby ensuring that the security mechanism remains active and the MCU continuously enforces its security policy. Any violation or failure to run security property checks will result in a HWD time-out, forcing a mandatory reset of the MCU or triggering some other remediating action (e.g., repairing the state). This guarantees that security properties are always periodically enforced on the system. In this section, we formalize HWD-based property enforcement.

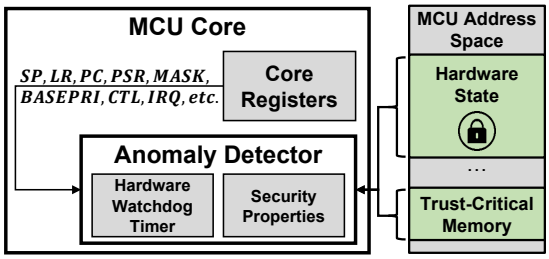

Fig. 2: Overview of HWD-assisted property enforcement.

A. HWD-Assisted Property Enforcement Overview

HWD-assisted property enforcement consists of several interlocking components, depicted by the block diagram in Figure 2. Let us consider the typical functionality of the MCU as it is booted and performs its intended task. As early as possible in the boot process (e.g., first-stage bootloader), a HWD timer is enabled. This always-on hardware timer must be periodically fed to avoid a mandatory reset. These timers are standard in low-end MCUs and cannot be disabled after activation [14]. The purpose of these timers is to maintain system stability by ensuring the MCU is responsive. However, in this context they are being leveraged to support security property enforcement via periodic checks. After configuring the HWD timer, a periodic non-maskable interrupt (NMI) is configured. The non-maskable status of this interrupt ensures it is promptly serviced and cannot be disabled or ignored. Within the interrupt handler, a set of pre-defined security properties for the MCU are checked to ensure they are satisfied. If satisfied, the NMI feeds the HWD timer, allowing the MCU to continue its intended tasks. If any property is violated, a warning is issued and remediating action is taken (e.g., repairing the MCU state or resetting the MCU).

The security properties evaluated in the NMI handler serve as a user-defined security policy for the MCU. These properties are logical statements that assess whether the system is in an allowable/benign security state based on the state of the MCU and its memory. This includes core registers (e.g., LR, PC), hardware configuration registers, and trust-critical memory regions. While these properties are determined per device based on the application, we provide a sample security property alongside an illustrative example in Section III-B.

The core functionality of HWD-assisted property enforcement can be represented with timing diagrams, commonly employed in work on RA, that illustrate the interactions between the prover (the MCU core) and the verifier (the HWD timer and property enforcement NMI). To outline the intended functionality, we discuss three cases, one where the MCU functions as intended and two where anomalous behavior is present in the device. In each case, we assume the MCU is already booted and running.

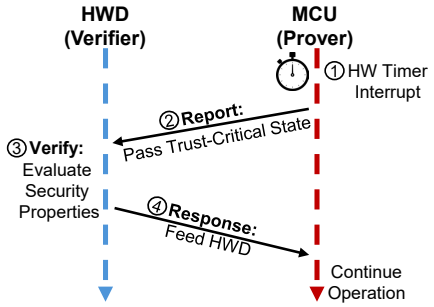

Fig. 3: Benign/intended behavior timing diagram for HWD-assisted property enforcement.

1) Case 1 – Intended Operation: This scenario considers the normal operation of the device and is depicted in Figure 3. The function of HWD-assisted property enforcement proceeds as follows. ①: a hardware timer triggers a periodic NMI requesting that the security properties be checked. ②: The MCU *reports* its current state by servicing the interrupt and ceding control to the handler. ③: The MCU state and memory contents are verified in the handler by checking the logical security properties. ④: If no violation of security properties is detected, the HWD timer is fed to ensure that the device continues operating. Control is then ceded back to the MCU to resume normal operation until the next report is requested.

2) Case 2 – Property Check Disabled: This scenario considers when the NMI is bypassed or disabled, preventing a security property check from occurring. This scenario is

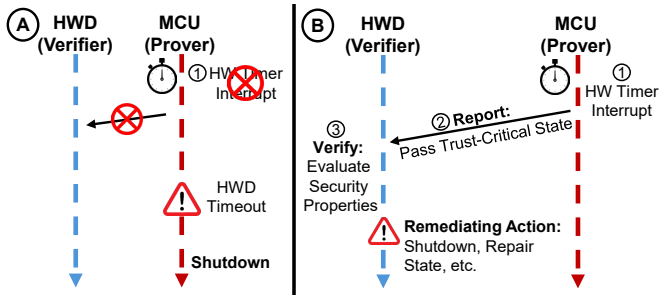

Fig. 4: Anomalous behavior timing diagram for HWD-assisted property enforcement.

depicted in Figure 4Ⓐ and described below. ①: Because the property-checking NMI was disabled, the *report* interrupt is never requested and the property-checking handler is never executed. This allows the device to continue operating without any security checking. However, in most MCUs, *the HWD timer cannot be disabled without resetting the device [14].* Thus, while property checking is disabled, the HWD timer is not. Since feeding the HWD timer is conditioned on all security properties being satisfied, the timer will not be fed. Once a timeout occurs, a mandatory device reset will be triggered and a violation (i.e., disabled property checking) will be reported to the user. Hence, the anomalous behavior will still be detected and remediating action will be taken even when the property checking is disabled.

3) Case 3 – Violated Security Property: This scenario considers when a security property is violated. It is depicted in Figure 4Ⓑ and described below. ①: A hardware timer triggers an NMI to check the security properties. ②: The MCU *reports* its current state by servicing the interrupt and ceding control to the handler. ③: During the security property check, a violation is detected. Instead of feeding the watchdog and ceding control as was done in Case 1, the user is notified of the violation and remediating action is taken (e.g., restoring the device to a secure state or resetting it).

B. Illustrative Example: Weak Readout Protection

We use a weak implementation of readout protection (RDP) discovered in STM32F0-series devices to illustrate the concepts underlying HWD-assisted property enforcement. RDP is employed in MCUs to protect system firmware against unauthorized reads or modifications. Prior work has demonstrated that weak RDP implementations in STM32F0 devices could be exploited to extract firmware in production devices [20]. Specifically, the authors demonstrated that fault injection could be used to exploit weak bit mappings, downgrading the RDP protection and allowing the firmware to be read.

HWD-assisted enforcement relies on a set of logical properties. For detection, the relevant security properties must first be formalized. For this example, we define our security property as: *"The RDP mode is always level 2"*. Enforcing this property at all times will protect the firmware from being read out. This property can be checked by ensuring two registers, $nRDP$ and RDP, always contain a specific value, defined by Eqn. 1.

$$(nRDP = 0x33) \wedge (RDP = 0xCC) \tag{1}$$

A NMI will be configured to periodically check this security property before feeding the HWD timer. Let us assume fault injection has been used to modify the RDP configuration as described by [20]. In this scenario, the periodic NMI will trigger a security property check. The handler for this NMI will identify that the RDP configuration has been modified, a violation of the security property. This will cause the user to be notified and initiate a mandatory device reset. We note that the time to detect such anamolous behavior is determined by the HWD timeout and the security property check frequency, which is set by the designer. Hence, the maximum time to detection is a configurable, designer-specified parameter. Additionally, notice that regardless of the mechanism that modifies the RDP configuration, ranging from hardware trojans to software exploits, the anomaly will still be detected and remediated, underscoring the generalizability of this approach.

C. Discussion of Merits and Limitations

We highlight three merits of the proposed approach. 1) HWD-assisted property enforcement provides a robust method for enforcing security properties without the need for custom hardware modifications, unlike prior work on RA [6], [8], [18], [19] and hardware property enforcement [10], [11]. 2) The proposed approach employs symptom-based detection, focusing on the impacts of potential security breaches on the MCU state rather than targeting specific vulnerabilities. This facilitates the detection of unknown vulnerabilities or security bugs, as highlighted by the illustrative example in Section III-B where modifications of RDP could be detected, regardless of their cause. 3) HWD-assisted property enforcement is generic, relying only on standard hardware components (i.e., a HWD timer) with no dependence on architecture or operating system. This allows for broad application across different platforms.

We identify three limitations: 1) HWD-assisted property enforcement can detect only threats that produce observable effects in MCU memory/state. Attacks that do not alter MCU state or only alter it transiently with no lasting effects will not be detected, allowing an attacker to clean up before property checks. However, property checks can not be masked and can be tuned to prevent such transient attacks. 2) Without privilege support, an adversary could disable property checking and feed the HWD timer directly. Mechanisms to mitigate this include using a trusted interrupt (e.g., secure interrupts in ARM TrustZone [21]) or requiring higher privilege to feed the timer. An off-chip HWD timer using a cryptographic nonce or device secret could also prevent unauthorized feeding. 3) HWD-assisted enforcement assumes the MCU was secure during production and the HWD timer is implemented properly.

IV. Case Study and Evaluation

To evaluate HWD-assisted property enforcement, we implemented it in an ARM Cortex-M4 (STM32L475) to enforce a

TABLE II: List of enforced security properties.

Security Property	Security Property Check	Relevant CWEs
1) Program counter remains in expected range	min address < SavedPC < max address	CWE-1260, CWE-506
2) Locked memory values are never modified	locked = saved (for all locked memory)	CWE-1231
3) Stack and heap spaces do not exceed given sizes	*CanaryLocation = CanaryValue	CWE-121, CWE-506
4) Sensitive info cleared on state transition	SensitiveData = ClearState if LowPowerFlag = 1	CWE-1272
5) Sensitive hardware configuration not changed after secure boot	Saved configuration = current configuration	CWE-1231, CWE-1233, CWE-1256
6) Debug mode is never enabled.	CoreDebug→CDEBUGEN = 0 (i.e., not set)	CWE-1191, CWE-1244, CWE-506, CWE-284, CWE-269
7) All secure boot stages run	Boot segment counter = 2 (i.e., expected number)	CWE-1274
8) All AES rounds run	Round count = 10-14 (i.e., expected number)	CWE-1240

set of 8 logical security properties based on the set of MITRE CWEs from the threat model in Section II-C. We use this implementation as a case study for evaluation.

A. Formalizing Security Properties

We developed a set of logical security properties based on the selected CWEs enumerated in Table I. These properties, outlined in Table II, will be enforced to detect anomalous behavior on the MCU. Each security property has a corresponding logical check that is implemented in the property enforcement NMI of the evaluation platform. This set of security properties is selected without the loss of generality. They are not intended to represent a comprehensive security policy, but rather to serve as a representative case study to evaluate the proposed approach and quantify its overheads. A variety of security policies for embedded systems, such as those proposed in [10], [22], could also be adopted by the HWD-assisted security mechanism proposed in this work.

B. Hardware Platform and Evaluation Methodology

A prototype implementation of HWD-based property enforcement was developed on an STM32L475, an ultra-low-power MCU with an ARM Cortex-M4. The device used X-CUBE-SBSFU for secure boot, a secure engine core, and a secure bootloader. FreeRTOS served as the OS. FreeRTOS served as the OS, running a sensor polling program that encrypted and transmitted data over UART. HWD-assisted property enforcement was implemented as described in Section III-A. The independent watchdog timer (IWDG) was configured with a 40ms timeout to serve as the HWD. General-purpose hardware timer 16 (TIM16) was configured to generate NMIs to trigger property checks. In the TIM16 interrupt handler, the security properties in Table II were checked. If no violations were found, the IWDG was fed. State information for property checking, as defined in Table II, was extracted from system registers, memory, and peripherals.

To evaluate the prototype platform, a set of benchmark anomalies were developed, each targeting a specific security property in Table II. Table III provides a brief description of each benchmark anomaly and the corresponding security property it violates. Each anomaly was produced using the MCU's debug interface to simulate the anomalous behavior and assess the performance of the prototype platform.

C. Experimental Results

We applied the 11 benchmark anomalies outlined in Table III to the STM32L475 prototype. Corresponding performance and overhead metrics were evaluated and are discussed below.

1) Anomaly Detection Rate: To assess the anomaly detection rate, the 11 benchmark anomalies were executed on the STM32L475 device 100 times. Each benchmark was launched in a separate trial without any synchronization to the current state of the MCU or the proposed solution. In all trials, the HWD-assisted property enforcement detected each of the 11 anomalies within 40ms. This is unsurprising because the maximum detection time is determined by the property checking interval (i.e., 40ms for this experiment). This interval can be configured by the designer based on application to improve maximum-time-to-detection at the expense of overhead. Additionally, because each benchmark was launched without any synchronization to MCU state, the detection time was uniformly distributed in the 40ms property checking interval.

2) False Detection Rate: To assess the false detection rate, the prototype device was operated for 500 hours (approximately 21 days). During this period, both the detector and user code were running to observe if any benign behavior would be flagged as anomalous. No anomalies were detected during this time. This is unsurprising as the detector checks logical properties that are not violated during normal operation.

3) Performance Overhead: The detector overhead was calculated by comparing the time spent on detector code to the time spent on other code. This was averaged over 1 hour. The detector took an average of 34 μs per 40ms HWD feed interval. Other user code accounted for the remainder of that interval. This results in a performance overhead under 0.1%.

4) Memory Overhead: Based on the prototype implementation, the memory (i.e., RAM) overhead for the detector was 5.1kB, which is under 4% of the available RAM on the device.

5) Evaluation Summary: The results of this case study support the efficacy of the proposed approach. All 11 anomaly detection benchmarks were successfully identified and no false detections were observed. Additionally, the overheads

TABLE III: Benchmark anomaly library for the evaluation of HWD-assisted property enforcement prototype platform.

#	Security Property	Test Description
1	Program counter remains in expected range	**Force Invalid PC Location:** Jump to an out-of-bounds function in a modified image to simulate code injection and improper execution.
2	Locked memory values are never modified	**Overwrite Locked Memory:** Overwrite locked memory addresses via JTAG to simulate an attack or data corruption.
3	Stack and heap spaces do not exceed given sizes	**Overwrite Stack Frame:** Perform stack buffer overflow, overrunning a stack buffer to modify the link register contents.
4	Stack and heap spaces do not exceed given sizes	**Overwrite Heap:** Allocate memory in the heap larger than available heap memory.
5	Sensitive info cleared on state transition	**Verify State Information:** A modified device image that removes clean-up methods for low-power mode transition.
6	Sensitive hardware configuration not changed after secure boot	**Overwrite Configuration Registers:** Modify various protected hardware timer configuration registers via JTAG.
7	Sensitive hardware configuration not changed after secure boot	**Fault Injection Attack:** Use electromagnetic fault injector to modify protected hardware state.
8	Debug mode is never enabled	**Activate Debug Mode:** Custom firmware image that leaves debug mode enabled.
9	Debug mode is never enabled	**Activate JTAG Interface:** Custom firmware image that enables JTAG interface.
10	All secure boot stages run	**Bypass Secure Boot Segment:** Modified secure boot that exits early from second-stage boot loader segment, failing to write segment complete flag.
11	All AES rounds run	**Early Exit from AES:** Custom TinyAES implementation that exits after 10 rounds with a 256-bit key.

remained well within the capabilities of low-end MCUs, supporting the practicality of the proposed approach.

V. Conclusion

In this work, we proposed HWD-assisted property enforcement for anomaly detection in low-end MCUs. This approach adapts existing custom-hardware-assisted trust mechanisms to utilize only standard on-chip hardware. The solution periodically checks user-defined security properties to detect anomalies. Security property checking is enforced through a HWD timer, which is standard in most low-end microprocessors, eliminating the need for additional hardware modifications. We evaluated this anomaly detection framework on an ARM Cortex-M4 device, successfully detecting all 11 anomaly benchmarks within 40ms. The method exhibited a small RAM overhead of 5.1kB and a performance overhead less than 0.1%.

VI. Acknowledgements

This work was supported by Eaton Corporation. The contents of this work do not necessarily reflect the position or policy of Eaton Corporation.

References

[1] A. Alwarafy *et al.*, "A survey on security and privacy issues in edge-computing-assisted internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4004–4022, 2020.

[2] L. Szekeres *et al.*, "Sok: Eternal war in memory," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 48–62.

[3] C. Cowan *et al.*, "Buffer overflows: Attacks and defenses for the vulnerability of the decade," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. IEEE, 2000.

[4] R. Roemer *et al.*, "Return-oriented programming: Systems, languages, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 1, pp. 1–34, 2012.

[5] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A systematic review of fault injection attacks on iot systems," *Electronics*, 2022.

[6] A. Caulfield, N. Rattanavipanon, and I. D. O. Nunes, "{ACFA}: Secure runtime auditing & guaranteed device healing via active control flow attestation," in *32nd USENIX Security Symposium*, 2023.

[7] L. Davi *et al.*, "Hafix: Hardware-assisted flow integrity extension," in *Design Automation Conference (DAC)*, 2015.

[8] I. D. O. Nunes, S. Jakkamsetti, and G. Tsudik, "Tiny-cfa: Minimalistic control-flow attestation using verified proofs of execution," in *Design, Automation & Test in Europe Conference & Exhibition*, 2021.

[9] B. Kuang *et al.*, "A survey of remote attestation in internet of things: Attacks, countermeasures, and prospects," *Computers & Security*, 2022.

[10] M. Hicks *et al.*, "Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 517–529.

[11] C. Sturton *et al.*, "Finalfilter: Asserting security properties of a processor at runtime," *IEEE Micro*, vol. 39, no. 4, pp. 35–42, 2019.

[12] T. Nyman *et al.*, "Hardscope: Thwarting dop with hardware-assisted run-time scope enforcement," *arXiv preprint arXiv:1705.10295*, 2017.

[13] C. Castelluccia *et al.*, "On the difficulty of software-based attestation of embedded devices," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 400–409.

[14] STMicroelectronics, *RM0351 ARM-Based MCUs Reference Manual*, STMicroelectronics, 2023.

[15] MITRE Corporation, "Common Weakness Enumeration (CWE)," 2024. [Online]. Available: https://cwe.mitre.org/

[16] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," in *Distributed Computing in Sensor Systems*. Springer, 2008, pp. 372–385.

[17] R. Kennell and L. H. Jamieson, "Establishing the genuinity of remote computer systems," in *12th USENIX Security Symposium*, 2003.

[18] X. Kovah *et al.*, "New results for timing-based attestation," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 239–253.

[19] D. Schellekens, B. Wyseur, and B. Preneel, "Remote attestation on legacy operating systems with trusted platform modules," *Science of Computer Programming*, vol. 74, no. 1-2, pp. 13–22, 2008.

[20] J. Obermaier and S. Tatschner, "Shedding too much light on a microcontroller's firmware protection," in *USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[21] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[22] V. Lotz, V. Kessler, and G. H. Walter, "A formal security model for microprocessor hardware," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 702–712, 2000.