NIST Cybersecurity White Paper
NIST CSWP 52

Firmware-Based Monitoring for Bus-Based Computer Systems

Guru Prasadh Venkataramani
Sanjay Rekhi
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.CSWP.52

April 15, 2026

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on 2026-03-20

How to Cite this NIST Technical Series Publication:
Venkataramani GP, Rekhi S (2026) Firmware-Based Monitoring for Bus-Based Computer Systems. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Cybersecurity White Paper (CSWP) NIST CSWP 52. https://doi.org/10.6028/NIST.CSWP.52

Author ORCID iDs
Guru Prasadh Venkataramani: 0000-0002-7084-7560
Sanjay Rekhi: 0009-0008-8711-4030

Contact Information
hwsec@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

Additional Information
Additional information about this publication is available at https://csrc.nist.gov/pubs/cswp/52/firmware-based-monitoring-for-bus-based-computer-s/final, including related content, potential updates, and document history.

All comments are subject to release under the Freedom of Information Act (FOIA).

Abstract

This paper describes design mechanisms that reconfigure component firmware as a network of forensic units that passively observe bus traffic to extract and share forensic data beyond typical communication. By employing consensus-building algorithms among these distributed units, the augmented firmware can collaboratively detect compromised nodes within a zero trust architecture to enable future system defense solutions.

Keywords

Attack detection; bus-based computing systems; distributed hardware systems; firmware; security forensics; system security monitoring.

Table of Contents

List of Tables

List of Figures

Executive Summary

The inherent vulnerabilities of bus-based computing systems that operate under implicit trust and the limitations of existing recovery methods due to a lack of detailed forensic data underline the need for more resilient and proactive system defense strategies. Tackling the ongoing risks associated with cascading implicit trust, especially in bus-based computing systems, calls for innovative solutions.

This paper describes a novel, low-cost approach to effectively monitoring security attacks in bus-based systems and designing mechanisms that can repurpose component firmware as a network of forensic units. These units enable a more comprehensive monitoring of system activity by providing observation vantage points for attacks that are not typically visible to OS- and application-level monitors. The system is built on distributed hardware components and uses consensus-building algorithms that can aggregate knowledge about a system's state from individual components to identify compromised nodes within a zero trust environment. It can also use indirect methods (e.g., event synthesis) to generate attack-related data for attack analyzer modules.

To minimize the system performance costs introduced by the firmware-based monitor, there is also a distributed local trust management mechanism enabled by a local trust table (LTT) at each node. The LTT tracks the trustworthiness of nearby nodes to enable fast, localized detection of malicious behavior without relying on expensive centralized monitoring methods. This consensus building helps validate the information held across individual nodes and improves an individual node's ability to borrow the system-wide knowledge collectively held by others. Finally, this hierarchical detection approach seeks to efficiently handle previously well-known threats within the node and resort to consensus-based detection when a local node does not have sufficient confidence in detecting the attack.

1. Introduction

Many critical computing systems, including those used by the Department of Defense, still rely on bus-based architectures [2][3]. These systems are complex systems-of-systems composed of modular components that implicitly trust one another [2]. These architectures are known to be vulnerable to cascading implicit trust attacks, where a single compromised component can jeopardize the entire system. Modern attackers have begun to operate beneath the OS to drop persistent malware, modify system behavior, and evade detection, all while undermining the very foundation of system integrity [4][5]. While such risks are well-documented, practical and deployable solutions remain elusive, especially for legacy systems.

Current forensic capabilities also struggle to meet the demands of modern cyber defense. They often lack the granularity and real-time insight needed for effective post-incident analysis and system recovery. For example, while file compromises may be detected, identifying the exact attack vector or reconstructing original file states is frequently hindered by limited forensic data.

Mitigating these challenges requires resilient, on-system detection that can help subsequent recovery mechanisms. Solutions must be both technically robust and practically viable to be implemented via firmware updates on existing hardware. This constraint underscores the immediate need for innovative approaches that directly address the risks of implicit trust and insufficient system state data.

2. Threat Model

Several security issues arise in bus-based computing systems. Potential threat vectors and system vulnerabilities could range from remote adversaries with ransomware that could corrupt a file system to hardware design-time bugs that could be triggered when an adversary exploits a known bug (e.g., hardware Trojan circuits). Security attacks manifest when such vulnerabilities are exploited by adversarial actors. A number of these attacks are not visible to OS layers and above, where typical system security monitors are currently implemented. Computer users must be aware of the vast landscape of attack types and the potential ways in which a bus-based computing system could be exploited. The proposed firmware-based monitoring infrastructure could help system administrators closely monitor individual hardware components and effectively close security loopholes.

Table 1 summarizes key threat types and provides real-world example scenarios in which they manifest as attacks.

Table 1. Threat types, descriptions, and real-world attack examples

Threat Type	Description	Example Real-World Attacks
Direct Memory Access (DMA)	Exploiting hardware to directly read or write to system memory, bypassing OS controls	Inception DMA attack using Peripheral Component Interconnect (PCI)-based devices to access system memory [6]
Ransomware	Malicious software that encrypts a victim's files through repeated file system writes	WannaCry malware [7]
Compromised Devices	Using authorized devices that are already infected and controlled by an attacker to infiltrate a system	NSA ANT catalog describes 'SURLYSPAWN' — a BIOS implant in PCI devices [8]
Lack of Isolation	Security failures due to insufficient separation between system components, allowing a breach in one area to impact others	Cloudbleed [9]
Denial of Service (DoS)	Overwhelming a system with traffic or requests, making it unavailable to legitimate users	A PCI-connected device floods the bus and overwhelms other components
Rootkits	Installing a rootkit (i.e., malicious software for privileged access) by exploiting the PCI bus interface	LoJax rootkit used PCI option ROM to persist below OS level [10]
Unauthorized Peripheral Access	Gaining access to a system or its data through peripherals without proper authorization	Thunderclap attack via Thunderbolt PCIe interface [11]
Physical Threats	Tampering with the physical hardware of a system to gain unauthorized access or control	Attacker plugs a malicious PCIe card into an unattended system

3. Proposed System Architecture

Figure 1 illustrates a high-level overview of the proposed system, where each bus-connected node has a lightweight local trust table (LTT) that stores the node's trust on other nodes.

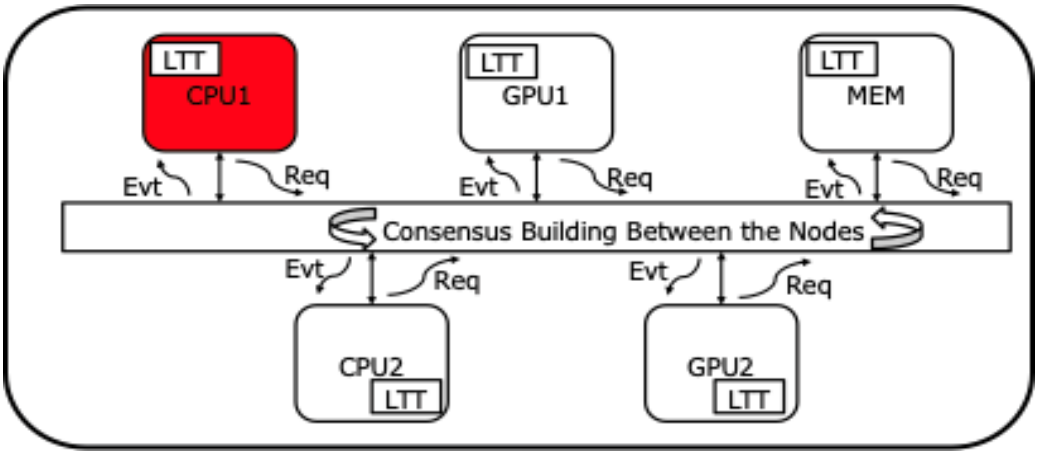

Fig. 1. A Zero-trust architecture with distributed consensus used to detect and repair compromised bus-connected nodes equipped with an LTT

A node will update its LTT based on the activities it observes on the bus. For example, during a ransomware attack, different system components would be invoked for various purposes (e.g., memory for file system activities, GPUs for throughput-oriented tasks). By monitoring these unusual activities within system-connected components, some nodes will update their LTT, and the firmware could indirectly infer the presence of ransomware attacks and carry out on-system recovery.

3.1. Challenges in Existing Approaches

Prior works [13] for compromised component detection have predominantly relied on either centralized event aggregation or dedicated monitoring mechanisms. This centralized approach aggregates system-wide events at a singular processing unit, leading to potential bottlenecks and scalability limitations [14]. Moreover, it introduces a single point of failure that can be exploited by attackers to bypass security mechanisms. Equally, dedicated monitors that continuously track system events often violate zero trust security models by assuming the trustworthiness of specific monitoring nodes. If these nodes are compromised, they themselves become attack vectors, thereby weakening the overall system security.

To address these limitations, this paper proposes a decentralized framework that employs data fusion techniques within the node to dynamically reconstruct event sequences. This approach ensures that the system can detect anomalies and potential compromises without introducing dependencies on specific monitoring points.

3.2. Firmware Instrumentation for Attack Detection

To effectively detect attacks at the firmware layer and identify compromised system components, low-cost instrumentation methods offer seamless integration advantages, especially in terms of minimizing the perturbation to sensitive system management layers like firmware. These methods enable the system to intelligently gather security-relevant event data while minimizing computational costs. Moreover, leveraging both direct and indirect data collection techniques provides a robust and adaptable solution for real-time attack detection.

3.2.1. Direct Event Collection From the System Bus (Preferred Method)

The most effective approach to gathering security-related data is through the direct observation of system events transmitted on the system bus. This method ensures high-fidelity data collection by capturing raw, unaltered event patterns (i.e., requests and responses) that can then be analyzed for signs of node compromise. Since all of the nodes inherently communicate over a shared bus, this instrumentation layer is designed to:

- **Monitor transactional data in real-time**, including memory access requests, I/O operations, and peripheral interactions
- **Track event patterns**, such as data exchanges, excessive memory reads/writes, and inter-node interactions
- **Detect deviations from normal event sequences** by leveraging predefined behavioral baselines

Embedding these monitoring capabilities within the firmware layer ensures that the system can passively observe and log security-critical events without significantly impacting performance or requiring extensive modifications to existing hardware architectures [15].

3.2.2. Indirect Event Collection Through Event Synthesis

If direct event collection is impractical (e.g., due to hardware constraints, limited bus access, or insufficient event visibility), indirect instrumentation techniques can be employed. One such method is hardware event synthesis [5], which allows the system to infer attack-related events by generating synthetic event data for each node's neighboring components. This approach involves:

- **Inferring missing data through logical correlations** using previously observed system-wide behaviors to predict likely attack signatures
- **Simulating attack-related activities** by reconstructing event sequences that match known malware behaviors at well-delineated time intervals, such as OS time quanta (typically 100 ms), thereby allowing the system to detect potential compromises even when direct data is unavailable

- **Leveraging neighbor-assisted detection** in which one node synthesizes attack-related data and shares it with adjacent nodes[1]

Enhancing the accuracy and effectiveness of event synthesis builds upon prior work in malware event synthesis, which demonstrated that Recurrent Neural Network (RNN)-based models can accurately learn and replicate malware-induced event patterns from pre-collected partial datasets logged by system monitors at the OS and hypervisor layers, particularly in the case of ransomware detection [12].

3.3. Detection Methods for Compromised Component Analysis

By observing system-wide events transmitted on the bus and correlating those observations across multiple connected nodes, this approach detects and infers malicious activities without relying on centralized monitoring or dedicated security modules that introduce single points of failure. This method also embraces zero trust principles by distributing security intelligence across the system and leveraging data fusion techniques for event reconstruction.

3.3.1. Local Data Fusion for Event Reconstruction

This detection methodology uses data fusion techniques to infer missing or compromised event data and to enable the system to reconstruct attack-related behaviors from indirect observations. Specifically, this approach involves:

- **Correlating multi-source event data:** Analyzing events across multiple bus-connected nodes can help identify inconsistencies or deviations that indicate malicious activity. For example, if a component's expected event sequence differs from observed patterns, it could signal compromise.
- **Reconstructing attack traces using historical patterns:** Memory access history and peripheral interaction logs can serve as behavioral baselines for detecting malware-induced deviations.
- **Employing statistical and machine learning models for anomaly detection:** Training predictive models on normal system behaviors can automate the detection of outliers and suspicious patterns that suggest potential breaches.

To enhance accuracy, baseline configurations must be established for various system parameters, including:

- Device and component configurations (e.g., expected DMA mappings, memory bindings)
- Normal system call sequences and access patterns
- Power consumption baselines for different operational states, which can be easier to map on components with a prior history of executing certain command sequences

[1] See Sec. 3.4.2 on how to build an LTT that guides the adoption of inputs from neighboring nodes.

Periodically verifying these baselines can help detect deviations that may indicate an ongoing attack. Moreover, comparing observed device states against historical baselines at specific time intervals (e.g., OS time quanta) can determine whether a component is exhibiting abnormal behavior suggestive of a compromise. Prior works [16] have proven that these analysis intervals are effective for system attack detection and defense.

Threshold-based detection can also ensure reliable malware prediction. These thresholds can be set empirically through observed attack behaviors or determined through formal analysis techniques. Potential signals for triggering detection logic include:

- **Excessive power usage:** Unexpected spikes in power consumption may indicate malware activity, particularly for cryptojacking or stealthy data exfiltration attempts.
- **Unauthorized memory access:** A compromised component may attempt to access restricted memory regions, signaling a potential intrusion.
- **Detection of unknown or unexpected devices:** If an unregistered device suddenly appears on the system bus, it may be an attacker's implant attempting to exfiltrate data or manipulate system behavior. Therefore, such unknown devices must be carefully vetted before they can be fully integrated into the system.

By dynamically adjusting these thresholds based on the system state and observed behaviors, this framework continuously adapts to evolving attack methodologies, ensuring long-term resilience against cybersecurity threats. This strategy also helps mitigate situations in which the system is continuously evolving due to hardware upgrades and/or software ecosystems being expanded with new applications.

This model can effectively detect malware behavior in certain scenarios, such as:

- **Memory access pattern analysis for ransomware detection:** Monitoring and analyzing memory access history can identify behavioral anomalies and the presence of ransomware, which often exhibits distinct memory access patterns (e.g., frequent file encryption attempts, unauthorized access to system-critical files).
- **GPU memory access correlation for keylogger detection:** A keylogger may attempt to intercept keystrokes by accessing the memory pages associated with keyboard buffers. Unauthorized keylogging activity can be inferred by logging GPU memory requests and cross-referencing them against keyboard buffer accesses.
- **Peripheral device interaction tracking:** Unusual interactions between connected devices (e.g., sudden spikes in data transfer between previously unrelated components) may indicate unauthorized data exfiltration attempts.

This method for detecting compromised components caused by malware marks a major improvement over traditional centralized or dedicated monitoring methods. Leveraging data fusion techniques and memory access correlations can help reconstruct event sequences and identify malicious activities with high accuracy. Additionally, formulating behavioral baselines and defining dynamic detection thresholds can enable the system to provide a robust and adaptive security solution aligned with zero trust principles.

3.4. Efficient Local Trust Management for Attack Monitoring

To avoid the performance costs associated with global system state monitoring, this architecture incorporates a distributed local trust management mechanism. At the core of this mechanism lies the LTT, which is a data structure maintained by each node within the system. The LTT serves as a localized repository of information pertaining to the trustworthiness of a node's immediate neighbors, which can be conveniently extended to all of the nodes in a bus-connected system. By focusing on local interactions and observations, the LTT enables the efficient and timely assessment of potential malicious activity without relying on centralized analysis, which can be computationally expensive. This section discusses immediate neighbors or local nodes for simplicity.

3.4.1. Structure and Content of the LTT

Node A is a direct neighbor of node B if B can observe A's behavior (or some of its behaviors) directly and indisputably. Here, "directly" means that the observed information (e.g., power dissipation, data transmission on the shared bus, memory bandwidth) does not come from other nodes, and "indisputably" means that A can identify the portion from B when the observation or information comes from multiple nodes.

Each entry within a node's LTT corresponds to one of its direct neighbors and stores information relevant to assessing that neighbor's trustworthiness. This information is dynamically updated based on locally observed behaviors and interactions. To derive a comprehensive understanding of a neighbor's reliability, an aggregate trust score synthesizes its value from various internal modules and performance indicators. Specifically, the trust score for a given neighbor will be influenced by:

- **Power Profile Analysis:** Deviations from a neighbor's baseline power consumption patterns can indicate malicious activity, such as excessive power drawn by a specific node (measured indirectly through monitoring the data requests via the bus transactions). Power consumption information can be obtained or inferred through integrated monitoring capabilities, indirect observations of bus activity and thermal characteristics, or modeling. The LTT will record and analyze the power consumption metrics and flag significant anomalies.
- **Performance Features:** Changes in a neighbor's expected performance metrics (e.g., volume of requests from the node, transaction latency, processing throughput) could be a symptom of resource exhaustion attacks or compromised software. In general, the amount and type of transaction activity that a chip generates on the shared bus can indicate its internal workload and performance. The LTT will track relevant performance indicators and incorporate deviations into the trust score calculation.
- **Memory Behavior:** Unusual memory access patterns, excessive memory footprint and/or usage, or attempts to access protected memory regions can signal malicious code execution. The LTT will monitor memory-related events and factor them into the trustworthiness assessment.

This proposed approach condenses all relevant information into a single numerical value (i.e., trust score) that represents the overall trustworthiness of a neighbor. While offering simplicity and ease of comparison, this representation might lose granular information about the specific types of suspicious behavior observed. If an aggregate score is not possible, a neighbor's trustworthiness can be represented as a vector of values, where each element captures an aggregate of a specific aspect of neighboring node's behavior (e.g., power consumption profile, memory-related characteristics, performance metrics).

A key characteristic of the LTT design is its *distributed* nature. Each node independently constructs and maintains its own LTT based exclusively on data that is locally accumulated and/or inferred. This approach ensures scalability and resilience since the trust management mechanism does not rely on a central authority or shared global state that could become a single point of failure. Nodes leverage their own sensor data, performance monitoring tools, and event logs to build a picture of their neighbors' behavior.

3.4.2. Enhancing Trustworthiness Through Cross-Validation

While the LTT is built on local observations, this methodology incorporates a consensus-building step that enhances the reliability of the trust assessments. Event data that is transmitted between nodes can be vetted for consistency through cross-validation. For example, if a memory module (MEM) on a particular node reports unusually high memory access patterns by its connected CPU, a third bus-connected component (e.g., GPU) can independently examine the event patterns that it observed on the shared bus to corroborate or contradict the MEM module's report. This mechanism allows individual components to validate suspicions and strengthen the evidence of potential compromises based on their own independent observations.

Crucially, the design of the LTT inherently aligns with the principles of zero trust since the information held in one node's LTT cannot directly contaminate the information maintained by other nodes. Isolating trust information prevents cascading failures or the propagation of false trust assessments across the system, ensuring that each node makes security decisions based on its own verified local context. By avoiding the implicit trust of neighboring entities and continuously verifying their behavior based on local evidence, the LTT design contributes to a robust and secure system.

The proposed detection methods here (see Sec. 3.3) also play a key role in reconstructing the internal state of a compromised component by observing the system-wide events transmitted on the system bus that connects the different components. These methods primarily rely on correlating observations from multiple bus-connected components to infer the behavior of potentially compromised nodes.

3.5. Hierarchical Detection

Figure 2 illustrates hierarchical detection in which event analysis modules are leveraged to invoke domain-specific threat detection mechanisms in a structured, multi-layered manner.

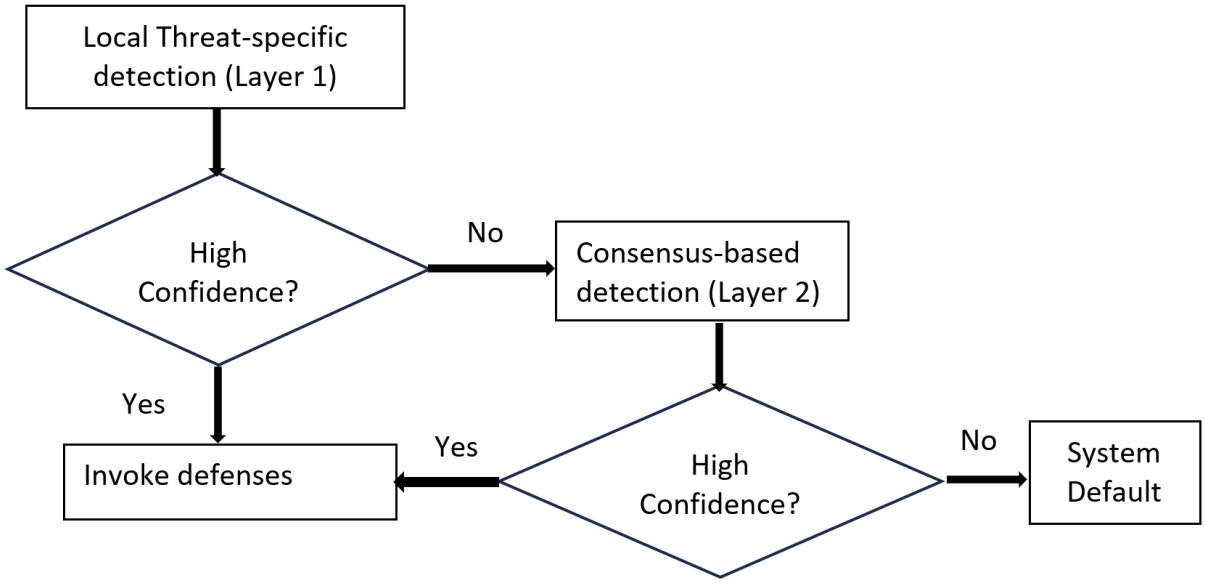

Fig. 2. Hierarchical defense in a distributed forensic firmware-enabled computing system

Threat-specific detection mechanisms serve as the first layer of defense, followed by anomaly detection as the final safeguard to identify previously unseen or evolving threats.

3.5.1. Threat-Specific Detection

The first layer of detection focuses on recognizing known attack patterns associated with specific threats, such as known ransomware signatures, characteristic file encryption activity, memory access patterns indicative of DMA attacks, or the presence of known rootkit signatures. Furthermore, it can look for patterns associated with unauthorized peripheral access or attempts to exploit buffer overflows. If such threats are detected with high confidence, firmware-supported repair mechanisms are immediately deployed. These countermeasures might include memory patching to counter DMA attacks, process isolation to contain ransomware, or system rollback to a clean state if a rootkit is suspected.

3.5.2. Consensus-Informed Detection

The second layer of detection becomes crucial when local detection lacks sufficient confidence, particularly in scenarios that involve compromised or rogue devices or attacks that exploit a lack of isolation. Neighboring hardware components can actively contribute to a threat assessment through the periodic monitoring of other nodes at pre-specified intervals. For example, if a node suspects that a rogue device is attempting a denial-of-service (DoS) attack by flooding the bus, its neighbors could corroborate this by observing similar patterns and exchanging their findings via consensus-building protocols [1]. Similarly, unusual bus activity from a device that is potentially engaged in a DMA attack or is exhibiting behavior inconsistent with its expected function could trigger alerts from other nodes.

3.5.3. Default

Finally, employing anomaly detection as default serves as the last line of defense against previously unseen or evolving threats, including sophisticated rootkit deployments, subtle unauthorized peripheral access, or even physical attacks that might lead to unexpected system states. By continuously analyzing execution patterns, resource usage, and system calls, this layer can identify deviations from established baselines. For example, an alert could be triggered by unusual memory access patterns after a physical attack or unexpected system call sequences indicative of a newly deployed rootkit.

4. Conclusions and Future Work

This paper describes instrumentation and detection methods for enhancing system security and resilience in bus-connected hardware components. The methods reconstruct the internal state of compromised components through smart analyses of system bus events and cross-component correlation that adheres strictly to zero trust principles. Data fusion techniques can be used for localized event reconstruction within each node to eliminate reliance on centralized aggregation. The proposed methods add value to existing system monitors by expanding their view below the OS layers and enabling future OS designs to leverage this improved security monitoring capability.

Future work (e.g., workshops, collaboration with industry partners) will investigate techniques that couple these firmware-based monitoring methods with system recovery mechanisms (e.g., self-healing techniques) as well as hardware-assisted designs that can perform firmware partitioning with fail-safe backups, actuator failsafe states, and secure over-the-air (OTA) firmware updates.

References

[1] Lamport L (2001) Paxos made simple. *ACM SIGACT News* (Distributed Computing Column) 32(4):51-58. https://doi.org/10.1145/568425.568433

[2] DARPA (2025) Reclaiming Bus-Based Systems During Compromise. DARPA Proposers Day 2025, Arlington, VA. Available at https://www.darpa.mil/sites/default/files/attachment/2025-01/darpa-presentation-red-c-proposers-day.pdf

[3] Keller J (2025) Military researchers eye future project for on-system cyber security in bus-based embedded computing. *Military & Aerospace Electronics*. Available at https://www.militaryaerospace.com/trusted-computing/article/55263709/cyber-security-for-bus-based-embedded-computing

[4] Hulme G (2021) Your Firmware Is a Wide Open Back Door. *Tanium.* Available at https://www.tanium.com/blog/your-firmware-is-a-wide-open-back-door/

[5] Vijayan J (2022) Researchers Discover Dangerous Firmware-Level Rootkit. Available at https://www.darkreading.com/threat-intelligence/researchers-uncover-dangerous-new-firmware-level-rootkit

[6] Alex M, Vargaftik S, Kupfer G, Pismeny B, Amit N, Morrison A, Tsafrir D (2021) Characterizing, exploiting, and detecting DMA code injection vulnerabilities in the presence of an IOMMU. *EuroSys '21: Proceedings of the Sixteenth European Conference on Computer Systems* (Association for Computing Machinery, United Kingdom), pp 395-409. https://doi.org/10.1145/3447786.3456249

[7] Chen Q, Bridges RA (2017) Automated Behavioral Analysis of Malware: A Case Study of WannaCry Ransomware. *16th IEEE International Conference on machine learning and applications* (ICMLA, Cancun, Mexico), pp 454-460. https://doi.org/10.1109/ICMLA.2017.0-119

[8] Yasin R (2015) 5 better tools to defend systems. Available at https://www.darkreading.com/threat-intelligence/the-nsa-playset-5-better-tools-to-defend-systems

[9] Gopstein D, Iannacone J, Yan Y, DeLong L, Zhuang Y, Yeh MKC, Cappos J (2017) Understanding misunderstandings in source code. *ESEC/FSE 2017, Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Association for Computing Machinary, Paderborn, Germany), pp 129-139. https://doi.org/10.1145/3106237.3106264

[10] Surve PP, Brodt O, Yampolskiy M, Elovici Y, Shabtai A (2023) SoK: Security Below the OS – A Security Analysis of UEFI. *arXiv preprint*. https://arxiv.org/abs/2311.03809

[11] Ruytenberg BLB (2022) When Lightning Strikes Thrice: Breaking Thunderbolt Security. Master's Thesis (Eindhoven University of Technology, Eindhoven, Netherlands). Available at https://research.tue.nl/en/studentTheses/when-lightning-strikes-thrice/

[12] Gogineni K, Derasari P, Venkataramani G (2022) Foreseer: Efficiently forecasting malware event series with long short-term memory. *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)* (IEEE, Storrs, CT), pp 97-108. https://doi.org/10.1109/SEED55351.2022.00016

[13] Fakhfakh F, Tounsi M, Mosbah M (2022) Cybersecurity attacks on CAN bus based vehicles: a review and open challenges. *Library Hi Tech* 40(5):1179-1203. https://doi.org/10.1108/LHT-01-2021-0013

[14] Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S (2013) On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News* 41(3):559-570. https://doi.org/10.1145/2508148.2485970

[15] Zhou L, Xiao J, Leach K, Weimer W, Zhang F, Wang G (2019) Nighthawk: Transparent System Introspection from Ring-3. *Computer Security – ESORICS 2019*, (Springer, Luxembourg City), *Lecture Notes in Computer Science* 11736, pp 217-238. https://doi.org/10.1007/978-3-030-29962-0_11

[16] Chen J, Venkataramani G (2014) CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware. *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (IEEE, Cambridge, UK). https://doi.org/10.1109/MICRO.2014.42