A Concise Review on Security and Privacy in Embedded Systems

Hemanth S[1], Jayanth G[2], Keerthana H[3], Kumaraswamy T[4], Dr. Govinda Raju M[5]
[1,2,3,4] *Dept. of Electronics and Communication Engineering RV College of Engineering Bengaluru, India*
[5]*Associate Professor Dept. of Electronics and Communication Engineering RV College of Engineering) Bengaluru, India*

Abstract—**Embedded systems face evolving security challenges due to their integration in IoT, automotive, and industrial appli- cations. These systems are vulnerable to various attacks targeting hardware, software, network communication, and cryptographic implementations.**

Hardware attacks include side-channel analysis and fault injection techniques which can extract cryptographic keys or by- pass authentication mechanisms. Software vulnerabilities involve buffer overflows, firmware modification, and secure boot exploits, enabling unauthorized access and persistent control. Network threats like Man-in-the-Middle (MITM) and replay attacks com- promise communication integrity, while cryptographic attacks, including brute-force and differential fault analysis (DFA), target embedded encryption systems.

To enhance security, compliance measures such as MISRA-C and CERT-C guidelines were evaluated in real-time operating systems (RTOS). CERT-C compliance, combined with machine learning techniques like Na¨ıve Bayes classification and Synthetic Minority Over-sampling Technique (SMOTE), improved vulnera- bility detection and reduced false positives. Hardware-based secu- rity mechanisms were also explored, including LHash-based jump integrity verification, FPGA-based access control used Hardware Isolation Mechanisms (HIMM), Physical Unclonable Functions (PUFs) were examined for secure cryptographic key generation, eliminating the risks of non-volatile key storage. Secure storage techniques such as Trusted Platform Modules (TPMs), Secure Elements (SEs), and flash memory encryption were also analyzed to protect sensitive data. These advancements highlight the need for hybrid security approaches integrating software resilience with real-time hardware monitoring to counter evolving threats in embedded systems.

Index Terms—**Side-channel Analysis, Fault Injection, MITM, Brute Force, DFA, MISRA-C, CERT-C, SMOTE, HIMM, PUF, TPM, Secure Elements**

I. INTRODUCTION

Embedded systems have undergone significant evolution, transitioning from simple microcontroller-based designs to complex, interconnected architectures that power modern IoT, automotive, medical, and industrial applications.

Early embedded systems were standalone, executing prede- fined tasks with minimal external interaction. However, ad- vancements in integrated circuits, real-time operating systems (RTOS), and wireless communication have transformed them into highly sophisticated, networked devices capable of real-time data processing and remote operation. The integration of AI, FPGA accelerators, and cloud connectivity has further enhanced their capabilities, making them integral to critical infrastructures.

With this evolution, security and privacy concerns have also increased. The increasing connectivity of embedded sys- tems exposes them to various cyber threats, including side- channel attacks, fault injection, firmware exploitation, and cryptographic vulnerabilities. Historically, security was not a primary design consideration, leaving many embedded devices susceptible to unauthorized access, data breaches, and system manipulation. As embedded systems are now widely deployed in safety-critical environments—such as autonomous vehicles, medical implants, and industrial control systems—securing them has become essential to prevent cyber-physical attacks that could lead to severe operational and financial consequences.

II. LITERATURE SURVEY

[1] This study evaluated the compliance of 16 open-source Real-Time Operating Systems (RTOSs) with MISRA-C:2012, a widely adopted secure coding standard. Using static code analysis (Cppcheck), the research assessed adherence to 153 rules, categorizing violations into mandatory, required, and advisory rules. Results showed that

while most RTOSs followed mandatory rules, numerous violations in required rules posed security risks. Common issues included type conversions, function declarations, loop constructs, and macro usage. The study emphasized the need for better adherence to MISRA-C to enhance the security and reliability of embedded systems.

[2] Improving Vulnerability Prediction Accuracy with CERT-C Secure Coding Standard This paper introduced a vulnerability prediction model based on CERT-C secure coding standards, aimed at detecting security flaws in software with greater accuracy. It addressed the cost inefficiency of traditional manual code audits by using machine learning techniques such as Na¨ıve Bayes and SMOTE to identify potential vulnerabilities. The study demonstrated that incorporating CERT-C violations as predictive features improved accuracy by over 5 percent compared to conventional models. Experiments conducted on Mozilla Firefox source code validated the approach's effectiveness in prioritizing security inspections and reducing false positives.

[3] This paper proposed a hardware-assisted monitoring framework to detect tampering with jump instructions and their target addresses in embedded systems. The method combined static analysis (offline) and dynamic monitoring (runtime) to ensure execution integrity. Using the LHash algorithm, the system generated integrity labels for jump instructions and monitored execution using a hardware security module. Implemented on an FPGA-based Xilinx Kintex-7 platform, the approach successfully detected Return-Oriented Programming (ROP) and Jump-Oriented Programming (JOP) attacks, providing real-time security without significant performance overhead.

[4] This research introduced a hardware-software co- design security framework to isolate FPGA hardware accelerators (IPs) from unauthorized access. The approach implemented Mandatory Access Control (MAC) policies similar to SELinux, extending access control to hardware IPs within embedded System-on-Chips (SoCs). A prototype demonstrated that the framework successfully prevented unauthorized access while maintaining low resource overhead. The study highlighted the growing security risks in FPGA-accelerated embedded systems and provided a scalable security model for protecting sensitive computations.

[5] This paper explored FPGA-based acceleration of OpenSSL cryptographic functions to enhance the performance of Transport Layer Security (TLS) protocols in embedded systems. It integrated hardware-accelerated implementations of AES-256 encryption, SHA-2 hashing, and RSA-2048 public-key cryptography into an FPGA-based embedded cryptosystem running Nios-II Linux RTOS. The system significantly improved SSL/TLS transaction speeds, reducing cryptographic overhead in networked applications. Results showed that hardware acceleration of cryptographic functions provided enhanced security and efficiency, making it suitable for applications such as secure communications, VoIP, and financial transactions.

[6] This paper explores security challenges in embedded devices, focusing on dynamic and static analysis techniques. It highlights the risks associated with firmware vulnerabilities, lack of security updates, and the increasing attack surface of embedded systems. The study emphasizes fuzzing and emulation-based security testing as effective methods for detecting software flaws. Key challenges include firmware complexity, lack of standard security frameworks, and difficulty in large-scale analysis. The paper provides insights into future security strategies, advocating for automated vulnerability detection and enhanced security-by-design approaches to protect embedded systems in IoT and critical applications.

[7] This paper examines fault injection attacks, where adversaries deliberately manipulate hardware execution conditions to exploit vulnerabilities in embedded software. It categorizes attacks into clock glitching, voltage fault injection, electromagnetic interference, and laser-based attacks, demonstrating how they can bypass security mechanisms such as cryptographic protections and authentication protocols. The study highlights how faults propagate through microprocessor execution, affecting software integrity. Additionally, it discusses countermeasures, including redundancy techniques, hardware-based anomaly detection, and secure coding practices to mitigate fault-based threats.

[8] This paper provides an overview of security risks in embedded systems, emphasizing

threats such as denial-of-service (DoS) attacks, lightweight cryptographic vulnerabilities, side-channel attacks, and insecure network protocols. It discusses challenges in implementing security within resource-constrained embedded devices, where performance and power limitations hinder the adoption of strong encryption and authentication mechanisms. The study also explores access control models, trusted platform modules (TPMs), and secure routing protocols to protect against cyber threats. It concludes by identifying research gaps in secure firmware updates, intrusion detection, and real-time security monitoring.

III. TYPES OF ATTACKS

A. Hardware Based Attacks

Hardware-based attacks target the physical components of embedded systems, often leveraging unintended behaviors or directly manipulating hardware to compromise security.Some of the most frequent and serious attacks are:

- Side-Channel Attacks (SCA): Side-channel attacks exploit unintentional information leakage from embedded devices, allowing attackers to extract sensitive data without directly breaking cryptographic algorithms.

- Power Analysis Attacks:

a) Simple Power Analysis (SPA): Observes power consumption patterns to deduce operations within the device.

b) Differential Power Analysis (DPA): Uses statistical analysis of multiple power traces to extract cryptographic keys.

- Electromagnetic Analysis Attacks: Similar to power analysis but relies on electromagnetic emissions from the device to infer secret data.

- Timing Attacks: Measure execution time variations to deduce sensitive information, particularly in cryptographic operations. Example: Power analysis attack successfully extracted cryptographic keys from a smart card used for authentication.

- Fault Injection Attacks: Fault injection attacks introduce intentional errors in embedded system operations, allowing attackers to manipulate behavior, bypass security features, or retrieve sensitive data.

- Laser Fault Injection: Focused laser beams alter transistor states, causing bit flips in sensitive regions of an integrated circuit (IC).

- Clock Glitching: Temporarily alters the clock signal, causing incorrect instruction execution and potentially exposing security flaws.

- Voltage Fault Injection: Manipulates power supply levels to induce computational errors and force the system into an unintended state. Example: Voltage glitching attack was used to bypass authentication on IoT microcontrollers, allowing unauthorized access.

- Reverse Engineering Attacks: Reverse engineering attacks involve extracting critical design information, firmware, or cryptographic keys from embedded devices.

- Decapsulation and Microscopy: Physical removal of chip packaging and analysis using techniques like electron microscopy to reconstruct circuit design and extract secrets.

- Code Extraction via JTAG/Debug Ports: Exploits debugging interfaces to gain low-level access, allowing attackers to dump firmware or modify execution. Example: Proprietary firmware was extracted from automotive ECUs, enabling attackers to manipulate vehicle control systems.

B. Software Based Attacks

Software-based attacks target vulnerabilities in firmware, operating systems, and applications, often exploiting weak se- curity mechanisms to gain unauthorized access or manipulate embedded system behavior.

- Buffer Overflow Attacks: Buffer overflow attacks occur when an attacker writes data beyond allocated memory boundaries, corrupting adjacent memory areas. This can overwrite function return addresses, hijacking execution flow to execute malicious code. Example: Attackers exploited firmware vulnerabilities to overwrite return addresses in IoT devices, enabling arbitrary code execution and system compromise.

- Code Injection Attacks: Code injection attacks insert malicious code into system memory, allowing attackers to execute unauthorized commands.

a) Shellcode Injection: Injects and executes shell com- mands to manipulate system behavior.
b) Firmware Modification: Alters embedded software to introduce backdoors or disable security mechanisms. Example: IoT cameras were compromised by injecting malicious firmware, granting remote attackers persistent control over video feeds and system settings.

- Malware and Rootkits: Malware and rootkits are embedded within firmware updates or applications, providing persistent access and control over compromised devices. These threats often evade detection by traditional security measures. Example: The Mirai botnet exploited weak default credentials in IoT devices, infecting thousands of embedded systems and using them for large-scale DDoS attacks.

- Exploiting Secure Boot Bypasses: Secure boot mechanisms are designed to prevent unauthorized firmware execution, but attackers can bypass these protections to load unsigned or malicious code. Example: A vulnerability in Tesla's secure boot mechanism allowed attackers to disable firmware integrity checks, enabling arbitrary firmware execution and potential system manipulation.

C. *Communication and Network Attacks*

Communication and network attacks exploit vulnerabilities in wired and wireless communication protocols used by em- bedded systems, enabling attackers to intercept, manipulate, or disrupt data transmission.

- Man-in-the-Middle (MitM) Attacks: MitM attacks occur when an attacker intercepts and alters communication between an embedded system and an external network, often by relaying or modifying transmitted data. Example: A Bluetooth MitM attack was used to intercept sensitive medical device data, potentially altering vital health parameters before reaching monitoring systems.

- Replay Attacks: Replay attacks involve capturing legitimate commands and retransmitting them to embedded systems, tricking devices into executing previously authenticated operations. Example: Car key fob replay attacks recorded and replayed wireless signals, allowing unauthorized unlocking of vehicles without direct key access.

- Spoofing Attacks: Spoofing attacks enable attackers to impersonate trusted devices, bypass authentication, and inject malicious commands. Example: A GPS spoofing attack on drones misled navigation systems, forcing UAVs to deviate from their intended flight paths, potentially leading to security breaches in critical missions.

- Jamming and Denial-of-Service (DoS) Attacks: Jamming and DoS attacks disrupt communication by overwhelming embedded networks with excessive signals or network requests, rendering devices unresponsive. Example: Wi-Fi jamming attacks targeted industrial embedded systems, causing operational downtime and disrupting automated processes.

D. *Cryptographic Attacks*

Cryptographic attacks target the encryption methods used in embedded systems, aiming to recover encryption keys, decrypt sensitive data, or weaken cryptographic security mechanisms.

- Brute-Force Attacks: Brute-force attacks systematically try all possible key combinations until the correct one is found. These attacks are particularly effective against systems with weak or short encryption keys. Example: Attackers exploited weak encryption in smart locks, allowing unauthorized access by systematically testing all possible passcodes.

- Differential Fault Analysis (DFA): DFA attacks introduce intentional faults during cryptographic computations to analyze output differences and infer secret keys. Example: A DFA attack on AES encryption in smart cards exploited induced computational errors, allowing attackers to extract encryption keys and decrypt sensitive data.

- Side-Channel Cryptographic Attacks: Side-channel attacks exploit unintended information leakage, such as power consumption, electromagnetic emissions, or execution timing variations, to recover cryptographic keys. Example: Power analysis attacks successfully extracted RSA private keys by monitoring power fluctuations during encryption operations.

IV. RECENT ATTACKS ON EMBEDDED SYSTEMS

Recent security attacks on embedded systems have targeted various devices globally, exploiting hardware and software vulnerabilities. The following mentioned incidents highlight the growing need for hardware-based security, encrypted com- munication, and real-time anomaly detection in embedded systems.

- VoltPillager Attack (2021):

A voltage fault injection attack targeting Intel SGX-enabled CPUs, bypassing secure enclave protections and allowing attackers to extract sensitive data.

- BrakTooth Vulnerabilities (2023):

A set of Bluetooth firmware flaws that enabled remote code execution and denial-of-service (DoS) attacks on embedded devices, including smartphones, IoT devices, and industrial systems.

- Tesla Jailbreak (2023):

A voltage glitching attack that bypassed Tesla’s secure boot mechanism, allowing unauthorized firmware modi- fications and full control over the vehicle’s systems.

- Medtronic Pacemaker Vulnerability (2024):

A wireless communication exploit that allowed remote attackers to modify pacemaker settings, posing life- threatening risks to patients.

- Qualcomm Modem Exploits (2024):

Firmware vulnerabilities in Qualcomm modem chips en- abled remote code execution on Android devices, allow- ing attackers to compromise mobile communications.

- BitForge Vulnerabilities (2023):

A cryptographic attack on Google Titan and Apple Secure Enclave chips, exposing private keys and compromising the security of secure elements in authentication systems.

- AMD Zenbleed Vulnerability (2023):

A side-channel attack exploiting speculative execution flaws in AMD Zen 2 CPUs, allowing attackers to extract sensitive data from registers.

- SPECTRE-BTI Attack (2024):

A branch target injection (BTI) exploit targeting modern Intel and AMD processors, bypassing previous Spectre mitigations to leak sensitive information.

- WiFi Dragonblood Attacks (2023-24):

A WPA3 security flaw allowing downgrade attacks and side-channel exploits, leading to WiFi password leaks and network intrusions.

- Hertzbleed Attack (2022-23):

A frequency-based side-channel attack on Intel and AMD CPUs, exploiting dynamic frequency scaling to leak cryptographic keys from embedded systems.

V. METHODOLOGY

A. *Hardware Solutions*

- Jump Instruction and Address Integrity using LHash Algorithm: This technique ensures that jump instructions and their target addresses in embedded systems are not tampered with.

a) Static Analysis: The system disassembles program code and applies the LHash algorithm to generate a unique static integrity label for each jump instruction and its target address.

b) Runtime Monitoring: A hardware module continuously monitors execution and calculates a dynamic integrity label using LHash.

c) Detection Mechanism: If the dynamic label does not match the static label, the system flags a security breach.

d) This baseline verification method ensures control-flow integrity and detects potential malicious code injections at the hardware level.

- FPGA-Based Security Using Hardware Isolation: This method enforces access control for FPGA-based hard- ware accelerators (IPs) through a two-layer validation system.

a) Hardware Isolation Mechanism (HIMM): Each IP has a hardware-level ”bouncer” (HIMM) that checks program IDs against a fast memory (AVC).

b) Software Integrity Monitoring Mechanism (SIMM): If an ID is not recognized, the request is validated through software, ensuring secure and efficient access control.

c) This mechanism prevents unauthorized access to hardware accelerators, making FPGA-based embedded systems more secure.

- RISC-V Security through Stack and Instruction Monitor- ing: RISC-V embedded systems employ hardware-based integrity monitoring to detect unauthorized modifications.

a) Instruction Integrity Monitoring: The system divides code into basic blocks and generates signatures (hashes) for them. At runtime, a Security Monitoring Unit (SMU) calculates and compares these signatures to detect modifications.

b) Function Return Address Monitoring (Shadow Stack): A duplicate ”shadow stack” stores return addresses separately. When a function returns, the system compares the expected address with the actual return address. Mismatch detection flags potential control-flow hijacking attacks.

c) This combined approach prevents hardware Trojans and return-oriented programming (ROP) attacks.

- Physical Unclonable Functions (PUFs) for Authentica- tion: PUFs utilize unique physical variations in semicon- ductor manufacturing to create unclonable cryptographic identities.

a) Challenge-Response Mechanism: A unique device challenge generates a distinct response, ensuring strong authentication.

b) On-the-Fly Key Generation: Instead of storing encryption keys in memory, PUFs generate cryptographic keys dynamically, reducing exposure to physical attacks.

c) PUFs enhance device authentication and cryptographic security in IoT, automotive, and industrial applications.

- Secure Storage Solutions for Embedded Systems: Sen- sitive data (encryption keys, credentials, and firmware) requires hardware-based secure storage.

a) Secure Elements (SEs): Tamper-resistant chips for secure key storage and encryption (used in IoT devices and smart cards).

b) Trusted Platform Module (TPM): Hardware cryptographic processors providing secure boot, encryption, and attestation.

c) Flash Memory Encryption: Protects stored data using hardware-accelerated encryption, ensuring confidentiality even if the storage medium is compromised.

d) Software-Based Encryption: Uses AES and key derivation functions (KDFs) with secure key storage mechanisms.

e) These storage solutions prevent unauthorized access, ensure long-term data integrity, and comply with privacy regulations (e.g., GDPR, HIPAA).

B. Software Solutions

- MISRA-C: The research conducted focused on analyzing Real-Time Operating Systems (RTOSs) for compliance with the MISRA-C coding standard. The study aimed to assess security vulnerabilities in open-source RTOS implementations and determine their adherence to secure coding practices. RTOSs are widely used in embedded and safety-critical systems where predictable execution and real-time performance are required. However, since many RTOSs are written in C, a language known for memory safety issues, following a secure coding standard is essential to prevent security vulnerabilities.

The research aimed to evaluate the MISRA-C:2012 compliance of 16 open-source RTOSs, identify common rule violations and their security implications, provide recommendations for safer RTOS development practices and MISRA-C was chosen because it is a widely recognized secure coding standard for the C programming language, particularly in automotive, industrial, and aerospace applications.

The study focused on 16 open-source RTOSs that met the following criteria:

a) Written in the C programming language

b) Open-source and publicly available

c) Actively maintained (updated in the last 5 years)

The RTOSs included in the study were: FreeRTOS, Zephyr, Apache NuttX, ChibiOS/RT, RIOT, CMSIS- RTOS, BeRTOS, QP/C, Phoenix-RTOS, Apache Mynewt, TinyOS, RTEMS, RT-Thread, Mbed OS, Contiki, LiteOS. and to evaluate MISRA-C compliance, the study used Cppcheck, a widely used static analysis tool (SAT).

The source code of each RTOS was obtained. The MISRA-C:2012 compliance checker was enabled. Cppcheck free version was used, which can check against 153 rules from MISRA-C:2012. It scanned the entire RTOS codebase for rule violations and the generated a report listing the violated MISRA-C rules along with the corresponding line numbers and file locations.

a) Total Lines of Code (LoC) Analyzed: 1,451,472 LoC.

b) Total MISRA-C Violations Detected: 100,143 violations.

c) Mandatory Rule Violations: 444 (0.44 percent)
d) Required Rule Violations: 56,129 (56.04 percent)
e) Advisory Rule Violations: 43,570 (43.50 percent)
f) RTOS with Most Violations: LiteOS (20 percent of its code violated MISRA-C).
g) RTOS with Least Violations: QP/C (2.74 percent violations).

Most Frequently Violated MISRA-C Rules: The analysis identified the top 10 most violated MISRA-C rules across the RTOSs.

a) Control Flow Issues Rule 15.5: (Advisory) Functions should have a single return point to reduce complexity. Rule 15.6: (Required) Compound statements must be used in if-else and loop bodies to prevent ambiguous execution.
b) Memory Safety and Type Conversions Rule 10.1: (Required) Operands of expressions should have matching essential types to prevent unexpected behavior.
Rule 10.3: (Required) Prevents assigning wider essential types to narrower types (e.g., double → int). Rule 10.4: (Required) Requires same essential type categories for arithmetic operations.
c) Pointer Management Issues Rule 11.4: (Advisory) Disallows pointer arithmetic between different types. Rule 11.5: (Advisory) Prevents conversion of void pointers to object pointers, reducing memory corruption risks.
d) Declaration and Definition Issues Rule 8.4: (Required) Ensures that external functions and objects have compatible declarations for safe linking. Rule 8.6: (Required) Prevents multiple identical identifier definitions, which can cause undefined behavior.
e) Loop Control Issues Rule 14.4: (Required) Requires loop conditions to be Boolean to prevent unexpected infinite loops.

- Key Findings and Security Implications:

a) Security Risks from MISRA-C Violations Memory Safety Issues: Buffer overflows, pointer mismanagement, and unsafe type conversions introduce critical vulnerabilities.

Undefined Behavior: Some RTOSs relied on ambiguous C constructs, making code execution unpredictable across different compilers.

Control Flow Weaknesses: Inconsistent return statements and poorly structured loops increase the risk of exploitable logical errors.

b) Recommendations for Improving RTOS Security Strict MISRA-C Compliance: Developers should enforce mandatory and required rules in all safety-critical embedded systems.

Enhanced Static Analysis: RTOS projects should adopt advanced SATs like LDRA, Coverity, or Klocwork for deeper security compliance checks.

Code Review and Audits: Regular manual reviews and automated security audits should be performed to catch violations early.

Memory Safety Enhancements: Use safe coding practices, such as bounds checking, avoiding raw pointer conversions, and enforcing type safety.

Secure Coding Training: Developers should be trained in MISRA-C and CERT-C guidelines to write more secure embedded software.

- CERT-C: This paper tells the approach for software vulnerability prediction based on CERT- C Secure Coding Standard (CCSCS) violation measures, it boosts up the accuracy and it is also cost-effective which identifies the security vulnerabilities by identifying the vulnerable code. Whereas the traditional predictions rely on software algorithm but does not provide accurate results.

The methodology tells about the following:

a) Target Software: Mozilla Firefox
b) Static Analysis Tool Used: LDRA Tool Suite (widely known as Liverpool Data Research Associates Tool Suite used for software verification and validation for static and dynamic analysis in security applications)
c) Machine Learning Algorithm: Na¨ıve Bayes Classifier (Probabilistic machine learning algorithm based on Bayes Theorem used for classification tasks in text classification and spam detection.)
d) Data Preprocessing Techniques: i) Chromosome- Based Data Representation (Gene-based classification of software attributes to encode problem variables.), ii)

Synthetic Minority Over-sampling Technique (SMOTE) to handle class imbalance, iii) Feature Selection using Information Gain (is a reduction technique to select nearly all relevant features and eliminate all irrelevant features.)

The major key finding is it explains about the following requirements:

a) The new CERT-C implementation beats the traditional implementation by over 5 percent
b) CERT-C violation usage improves the Probability of Detection (fraction of nominal discontinuity)
c) False positive rates (measures the accuracy of a test.) found to be slightly higher in CERT-C models.
d) Combination of CERT-C violations and traditional software metrics gives highest prediction accuracy.
e) The conclusion majorly provides the three prominent features:
i) CERT-C Secure Coding Violations usage drastically improves the vulnerability prediction models
ii) Combination of CERT-C rules and traditional software gives the better results
iii) SMOTE which is a machine learning technique helps in maintaining class distribution equally and also helps in increasing the accuracy.

VI. CONCLUSION

This study provided an in-depth analysis of security threats and mitigation strategies in embedded systems, with a particular focus on hardware and software security solutions. The evaluation of MISRA-C and CERT-C compliance in Real-Time Operating Systems (RTOSs) revealed critical vulnerabilities stemming from insecure coding practices, improper memory management, and weak access controls. The findings emphasized the importance of adhering to secure coding standards, utilizing static analysis tools, and integrating machine learning-based vulnerability prediction models to enhance software security.

Furthermore, the research explored hardware-assisted security techniques, including LHash-based jump integrity verification, FPGA-based hardware isolation, Physical Unclonable Functions (PUFs), and Trusted Platform Modules (TPMs). These mechanisms were shown to effectively mitigate firmware tampering, unauthorized access, and cryptographic key extraction attacks. The study demonstrated that a hybrid security approach, combining software resilience with real- time hardware monitoring, is essential for countering modern cyber threats in IoT, automotive, medical, and industrial embedded systems.

REFERENCES

[1] X. Zhou, P. Wang, L. Zhou, P. Xun, and K. Lu, “A Survey of the Security Analysis of Embedded Devices,” Sensors, vol. 23, no. 9221, pp. 1-18, Nov. 2023.

[2] B. Yuce, P. Schaumont, and M. Witteman, “Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation,” J. Hardw. Syst. Secur., vol. 2, pp. 111-130, 2018.

[3] K. Fysarakis, G. Hatzivasilis, K. Rantos, A. Papanikolaou, and C. Manifavas, “Embedded Systems Security Challenges,” in Proc. 4th Int. Conf. Pervasive Embedded Comput. Commun. Syst. (PECCS), 2014,
pp. 255-266.

[4] J. Yang, D. Ryu, and J. Baik, “Improving Vulnerability Prediction Accuracy with Secure Coding Standard Violation Measures,” in Proc. IEEE BigComp, 2016, pp. 115-122.

[5] J. Song, R. Shigdel, A. Pokharel, and J. Alves-Foss, “Real-Time Operating Systems’ Compliance With MISRA-C Coding Standard: A Comprehensive Study,” IEEE Access, vol. 12, pp. 151955-151970, 2024.

[6] Q. Hao, Z. Zhang, T. Le, J. Zhang, J. Ma, J. Wang, J. Liu, and X. Wang, “A Hardware-Assisted Security Monitoring Method for Jump Instruction and Jump Address in Embedded Systems,” in Proc. IEEE ICNISC, 2022.

[7] S. K. Saha and C. Bobda, “FPGA Accelerated Embedded System Security Through Hardware Isolation,” in Proc. AsianHOST, 2020.

[8] M. Khalil-Hani, V. P. Nambiar, and M. N. Marsono, “Hardware Acceleration of OpenSSL Cryptographic Functions for High-Performance Internet Security,” in Proc. IEEE Int. Conf. Intell. Syst. Model. Simul. (ISMS), 2010, pp. 374-380.