MANTECH Publications
International Journal of Embedded Electrical Systems and Automation
Volume 1, Issue 2, July-December 2025

Security and Trustworthy Embedded Systems

Sukanya Rawat[1], Ravi S. Sharma[2], Nisha Patil[3], Arjun Deshmukh[4]
Students[1, 2, 3], Professor[4]
Department of Electronics and Communication Engineering
Arya College of Engineering
Email ID: *sukanyarawat92@gmail.com[1], sharma1ravi@yahoo.com[2], nishapati20l@rediffmail.com[3]*

ABSTRACT

Embedded systems are increasingly becoming the backbone of modern digital infrastructure, ranging from medical devices and automotive systems to industrial control and IoT devices. With this pervasive integration, security and trustworthiness have emerged as crucial concerns, as attacks on embedded devices can have severe consequences, including financial loss, privacy breaches, and even threats to human safety. This paper provides a comprehensive review of the security challenges faced by embedded systems, the threats targeting them, and strategies for ensuring trustworthiness. It discusses hardware-based, software-based, and hybrid security mechanisms, including encryption, secure boot, hardware root-of-trust, and intrusion detection systems. Additionally, the paper examines evaluation metrics for trustworthy systems, challenges in real-time embedded environments, and emerging trends such as blockchain integration and AI-based anomaly detection. The aim is to provide a roadmap for designing resilient and reliable embedded systems for diverse applications.

KEYWORDS: *Embedded systems, security, trustworthiness, IoT, secure boot, hardware root-of-trust, intrusion detection, and encryption.*

INTRODUCTION

Embedded systems are specialized computing systems that perform dedicated functions within larger systems. They are widely deployed in industrial control, automotive electronics, medical devices, aerospace, and smart home applications. Unlike general-purpose computers,

embedded systems often have constrained resources such as limited processing power, memory, and energy capacity, which make traditional security approaches challenging to implement.

With the expansion of IoT (Internet of Things) and Cyber-Physical Systems (CPS), embedded devices are increasingly connected to networks, exposing them to external attacks. Malicious intrusions, side-channel attacks, firmware tampering, and data leakage pose significant risks to their functionality and integrity. As a result, **trustworthiness** — the ability of a system to behave as expected, securely and reliably — has become an essential requirement.

The main objective of this paper is to review the current security mechanisms in embedded systems, highlight their limitations, and propose directions for ensuring robust and trustworthy designs.

SECURITY CHALLENGES IN EMBEDDED SYSTEMS

Embedded systems, ranging from IoT devices to industrial controllers, operate under specific constraints that make securing them uniquely challenging. Unlike general-purpose computers, these systems must balance security with performance, cost, and power efficiency. Key security challenges include:

1. Resource Constraints

- **Explanation:** Embedded devices often have limited CPU power, memory, and storage. Implementing strong security protocols like AES-256 encryption, TLS with mutual authentication, or full-featured intrusion detection systems can overwhelm these resources.
- **Implications:** Security measures must be lightweight, often compromising robustness. For example, IoT sensors may use simplified cryptography, making them more vulnerable to brute-force or cryptanalysis attacks.

Mitigation Approaches:

- Use hardware-accelerated cryptographic modules.
- Employ lightweight cryptographic algorithms designed for constrained devices (e.g., SPECK, SIMON).
- Optimize firmware to balance security with performance.

2. Physical Accessibility

- **Explanation:** Many embedded systems are deployed in unmonitored or public locations, such as ATMs, smart meters, industrial sensors, or medical devices. This physical exposure makes them prone to tampering or theft.

Threats Include:

- **Tampering attacks:** Manipulating hardware to alter behavior.
- **Reverse engineering:** Extracting firmware or secrets from the device.
- **Side-channel attacks:** Exploiting information leaked via power consumption, timing, or electromagnetic emissions.

Mitigation Approaches:

- Physical security mechanisms like tamper-evident seals or hardened enclosures.
- Obfuscation of firmware and encryption of critical data.
- Implementation of side-channel resistant designs.

3. Network Vulnerabilities

- **Explanation:** Many embedded devices are network-connected (IoT, industrial control, smart homes). Connectivity opens them to remote attacks.

Common Network Threats:

- **Denial of Service (DoS):** Overloading the device or network to disrupt operation.
- **Man-in-the-Middle (MITM):** Intercepting and altering communication between devices.
- **Spoofing/Impersonation:** Falsely representing a device or user to gain unauthorized access.
- **Malware Injection:** Uploading malicious firmware or code to compromise the device.

Mitigation Approaches:

- Secure communication protocols (TLS/DTLS).
- Device authentication and identity verification.
- Network monitoring for anomalous behavior.

4. Software Vulnerabilities

- **Explanation:** Firmware and software in embedded systems can contain bugs or security flaws that attackers exploit. Updating software is often difficult due to the deployment scale, connectivity limitations, or safety-critical operation.

Examples:

- Buffer overflows in device firmware.
- Hardcoded passwords in IoT devices.
- Outdated libraries with known vulnerabilities.

Mitigation Approaches:

- Secure coding practices.
- Over-the-air (OTA) update mechanisms with authentication.
- Vulnerability scanning and patch management strategies.

5. Real-Time Requirements

Explanation:

Some embedded systems (e.g., automotive controllers, medical devices, industrial robots) require deterministic timing and low-latency operation. Introducing security mechanisms may delay responses, causing safety hazards or system failure.

Challenges:

- Balancing encryption, authentication, or logging with real-time deadlines.
- Ensuring that security overhead does not compromise safety or reliability.

Mitigation Approaches:

- Use hardware-based security accelerators.
- Implement security policies tailored to critical paths only.
- Prioritize essential operations over less critical security functions in real-time scenarios.

Table 1. Major Security Challenges in Embedded Systems

Challenge	Description	Impact on Security
Resource Constraints	Limited CPU, memory, and energy	Restricts use of strong encryption and detection
Physical Accessibility	Devices can be physically accessed	Vulnerable to tampering, side-channel attacks
Network Vulnerabilities	Exposure to internet or LAN	Risk of DoS, MITM, malware
Software Vulnerabilities	Bugs or insecure firmware	Exploitation leads to control loss
Real-Time Requirements	Timing constraints for critical operations	Security measures must be lightweight

THREATS TARGETING EMBEDDED SYSTEMS

Embedded systems are vulnerable to a wide range of threats due to their combination of hardware and software components, often deployed in physically exposed and network-connected environments. These threats can be broadly classified into **hardware-level** and **software-level** threats.

1. Hardware-Level Threats

Hardware threats exploit the physical or architectural aspects of embedded systems.

a) Side-Channel Attacks

- **Explanation:** Attackers gather information from physical signals emitted by a device during operation, such as power consumption, electromagnetic radiation, or timing variations.

Examples:

- Power analysis to extract cryptographic keys from smart cards.
- Timing attacks on embedded processors to infer secret data.
- **Impact:** Can lead to full system compromise without exploiting software vulnerabilities.

Mitigation:

- Masking or randomizing operations.
- Shielding circuits and sensors.
- Using side-channel resistant cryptographic algorithms.

b) Fault Injection Attacks

- **Explanation:** Attackers deliberately induce faults in the hardware to disrupt normal operation and bypass security checks.

Methods:

- Voltage spikes, clock glitches, or electromagnetic interference.
- Laser or focused energy attacks on microchips.
- **Impact:** Can skip authentication, corrupt data, or trigger unsafe behavior.

Mitigation:

- Implementing redundancy and error-checking circuits.
- Using sensors to detect abnormal operating conditions.

c) Hardware Trojans

- **Explanation:** Malicious modifications introduced during the design or manufacturing of integrated circuits.

Types:

- Backdoors that allow unauthorized access.
- Triggered logic bombs that activate under specific conditions.
- **Impact:** Can remain dormant until exploitation, compromising security and safety.

Mitigation:

- Rigorous supply chain verification.
- Post-manufacturing testing and hardware attestation.

2. Software-Level Threats

Software threats exploit vulnerabilities in the device firmware, operating system, or application Logic.

a) Malware and Ransomware

- **Explanation:** Embedded devices with network connectivity can be targeted by malware that disrupts operation, steals data, or demands ransom.

Examples:

- IoT botnets used in DDoS attacks.
- Malicious code targeting industrial controllers or smart home devices.
- **Impact:** Loss of control, data breaches, or operational disruption.

Mitigation:

- Firewalls, antivirus for embedded OS, and secure boot mechanisms.
- Network segmentation and anomaly detection systems.

b) Firmware Tampering

- **Explanation:** Unauthorized modification or replacement of firmware changes device behavior.

Examples:

- Adding hidden data collection routines.
- Altering control logic in industrial systems.
- **Impact:** Can bypass security controls, corrupt data, or enable backdoor access.

Mitigation:

- Digitally signed firmware and secure update protocols.
- Integrity checks and rollback mechanisms.

c) Buffer Overflows and Code Injection

- **Explanation:** Poorly written software can allow attackers to overwrite memory or inject malicious code.

Examples:

- Exploiting input validation errors in embedded web servers.
- Command injection in IoT devices with insecure APIs.

- **Impact:** Full control of the device, data exfiltration, or persistent malware installation.

Mitigation:

- Secure coding practices and static/dynamic code analysis.
- Runtime memory protection and input validation mechanisms.

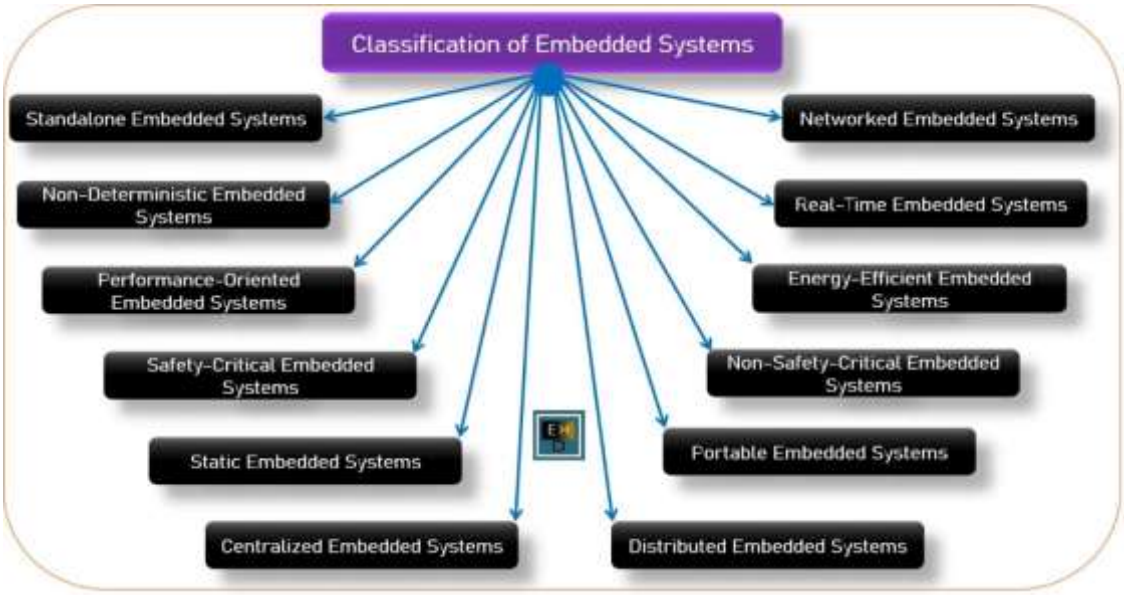

Figure 1. Threat Categories for Embedded Systems

SECURITY MECHANISMS FOR EMBEDDED SYSTEMS

Embedded systems require a combination of **hardware, software, and hybrid mechanisms** to ensure security, confidentiality, integrity, and availability. These mechanisms are designed to protect against both physical and logical attacks while maintaining the operational constraints of embedded devices.

1. Hardware-Based Security

Hardware-based mechanisms leverage physical components to enforce trust and protect sensitive operations.

a) Secure Boot

- **Explanation:** Ensures that the device boots only using firmware and software verified as authentic and untampered.

- **How it works:**
  - The bootloader verifies the digital signature of firmware before execution.
  - If verification fails, the device refuses to boot, preventing malicious code from running.

- **Example:**
  - Modern ARM-based IoT devices use secure boot to prevent malware from persisting through reboots.

- **Benefits:**
  - Prevents persistent malware.
  - Establishes a chain of trust from hardware to OS and applications.

b) Hardware Root-of-Trust (RoT)

- **Explanation:** A dedicated hardware module securely stores cryptographic keys, certificates, and secrets. It acts as a trusted anchor for verifying firmware and software integrity.

- **Example:**
  TPM (Trusted Platform Module) or HSM (Hardware Security Module) embedded in smart meters or automotive ECUs.

- **Benefits:**
  - Protects sensitive cryptographic material from extraction.
  - Enables secure authentication and verification processes.

c) Trusted Execution Environments (TEE)

- **Explanation:**
  Provides isolated, secure environments within the processor to execute sensitive operations (e.g., cryptographic computations, key management).

- **Example:**
  ARM TrustZone creates a secure "world" separate from the normal operating system for critical tasks.

- **Benefits:**
  - Protects critical code and data even if the main OS is compromised.
  - Enables secure processing of sensitive information like biometric data or payment credentials.

2. Software-Based Security

Software mechanisms focus on protecting communication, monitoring activity, and controlling access.

a) Encryption Algorithms

- **Explanation:** Encrypts data stored on the device or transmitted over networks.

- **Types:**
  - Lightweight symmetric algorithms (AES-128, ChaCha20) for constrained devices.
  - Asymmetric algorithms (ECC, RSA) for secure key exchange and authentication.

- **Benefits:**
  - Confidentiality and integrity of data.
  - Prevents eavesdropping and unauthorized access.

b) Intrusion Detection Systems (IDS)

- **Explanation:** Monitors device or network activity for abnormal behavior indicative of attacks.

- **Example:**
  - Signature-based IDS for known malware patterns.
  - Anomaly-based IDS using machine learning to detect deviations from normal device behavior.

- **Benefits:**
  - Real-time detection of attacks.
  - Can trigger alerts or defensive actions automatically.

c) Access Control Policies

- **Explanation:** Defines which users, processes, or devices can access specific functions or data.

- **Types:**
  - Role-based access control (RBAC) for device functions.
  - Attribute-based access control (ABAC) for more granular security.

- **Benefits:**
  - Prevents unauthorized access to sensitive operations.
  - Limits potential attack surface.

3. Hybrid Security Approaches

Hybrid approaches combine hardware and software mechanisms to provide layered defense.

- **Explanation:**
  Using hardware-based trust anchors with software-based monitoring creates stronger security than either approach alone.

- **Example:**
  - Secure boot with a hardware Root-of-Trust ensures only verified firmware runs.
  - An IDS monitors the system at runtime to detect unexpected behavior even after boot.
  - **Benefits:**
  - Protects against both static (boot-time) and dynamic (runtime) attacks.
  - Enhances resilience against sophisticated attacks that target multiple system layers.

TRUSTWORTHY EMBEDDED SYSTEMS

Trustworthiness in embedded systems goes beyond mere security. It encompasses **reliability, safety, and privacy**, ensuring that devices operate correctly, safely, and responsibly, even in adverse conditions. This is crucial for systems deployed in critical domains like healthcare, automotive, industrial automation, and smart infrastructure.

1. Reliability

- **Reliability**

  Ensures that an embedded system performs its intended functions consistently, even under stress, over time, and in varying environmental conditions.

- **Techniques for Reliability**
  - **Redundancy:** Deploying multiple components (hardware or software) so that failure of one does not compromise system operation.
  - Example: Dual modular redundancy (DMR) or triple modular redundancy (TMR) in flight control systems.

- **Error Detection and Correction**
  - Using error-correcting codes (ECC) or parity checks to detect and correct data corruption in memory or communication.
  - Example: ECC memory in industrial controllers

- **Fault-Tolerant Design**
  - Designing systems to continue functioning despite component failures.
  - Example: Automotive braking systems use sensors in parallel to ensure proper operation if one fails.

- **Benefits:**
  - Minimizes downtime and operational disruption.
  - Increases confidence in mission-critical systems.

2. Safety

Safety ensures that embedded systems do not cause harm to users, the environment, or other systems, even in the event of failure. Safety is especially critical for **life-critical or mission-critical applications**.

Safety Mechanisms:

- **Watchdog Timers:**

  Hardware or software timers that reset the system if it becomes unresponsive or behaves

unexpectedly.

- Example: Used in pacemakers and automotive controllers.

- **Real-Time Monitoring:**
  - Continuous observation of system parameters to detect anomalies or hazardous conditions.
  - **Example:** Industrial robots monitor torque and position sensors to prevent collisions.

- **Fail-Safe Designs:**
  - Designing systems to default to a safe state during failure.
  - **Example:** Elevator controllers stop movement if sensors fail.

- **Safety Standards Compliance:**
  - Adhering to standards such as ISO 26262 (automotive) or IEC 61508 (industrial) ensures safety-critical practices.
  - **Benefits**:
    - Prevents accidents, injuries, or damage.
    - Builds trust for adoption in sensitive domains.

3. Privacy

Privacy ensures that sensitive or personal data handled by embedded systems is protected against unauthorized access or misuse. This is critical for IoT, healthcare devices, smart homes, and wearable technology.

- **Privacy Protection Techniques:**

a) **Data Encryption:**
  - Encrypt sensitive data at rest and in transit using symmetric or asymmetric cryptography.
  - Example: Smart home cameras encrypt video streams before transmission.

b) **Secure Storage:**
  - Store sensitive information (e.g., medical records, biometric data) in secure, tamper-proof memory.

- Example: Trusted Platform Modules (TPM) or hardware-secured flash memory.

c) **Access Control:**

- Implement policies that restrict which users or processes can access sensitive data.
- Example: Role-based or attribute-based access control in connected medical devices.

d) **Data Minimization:**

- Collect and store only necessary information to reduce exposure.
- Example: Smart meters anonymize user energy usage data before storage.
- **Benefits:**
  - Protects user trust and compliance with privacy regulations (e.g., GDPR, HIPAA).
  - Reduces risk of identity theft, data leaks, or unauthorized surveillance.

Figure 2. Components of Trustworthiness in Embedded Systems

EMERGING TRENDS IN EMBEDDED SYSTEM SECURITY

1. AI-Based Security

Machine learning algorithms are increasingly applied to detect anomalous patterns in embedded system behavior, improving intrusion detection and adaptive response.

2. Blockchain Integration

Blockchain can be used for secure firmware updates and trusted communication between devices in IoT networks.

3. Physically Unclonable Functions (PUFs)

PUFs exploit manufacturing variations to generate unique device identifiers, improving authentication and key storage without extra hardware.

4. Lightweight Cryptography

With resource-constrained devices, algorithms such as SPECK, SIMON, and PRESENT are gaining popularity.

CHALLENGES AND FUTURE DIRECTIONS

Despite advances, several challenges remain:

- **Balancing Security and Performance:** Lightweight mechanisms may compromise security; heavyweight methods may reduce efficiency.
- **Standardization:** Diverse embedded systems require common security frameworks.
- **Firmware Update Management:** Secure and reliable OTA updates remain challenging.
- **AI Adoption Risks:** ML models themselves can be attacked (adversarial attacks).

Future research should focus on **adaptive security**, **energy-efficient cryptography**, and **resilient architectures** capable of surviving both cyber and physical attacks.

CONCLUSION

Security and trustworthiness are critical requirements for modern embedded systems due to their pervasive deployment and exposure to threats. This paper reviewed the main security challenges, threats, and mechanisms for embedded systems, highlighting the need for hardware, software, and hybrid solutions. Trustworthiness encompasses not only security but also reliability, safety, and privacy. Emerging trends such as AI-based intrusion detection, blockchain integration, and lightweight cryptography show promise but introduce new challenges. Addressing these issues is crucial to ensure the safe and reliable operation of embedded systems in diverse domains.

REFERENCES

1. S. Ravi, A. Raghunathan, P. Kocher, S. Hattangady, “Security in Embedded Systems: Design Challenges,” *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 461–491, 2004.

2. Halperin et al., “Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses,” *IEEE Security & Privacy*, vol. 7, no. 3, pp. 30–40, 2009.
3. Boztas et al., “Lightweight Cryptography for Embedded Devices,” *Journal of Cryptographic Engineering*, vol. 5, pp. 101–112, 2015.
4. P. Kocher, J. Jaffe, B. Jun, “Differential Power Analysis,” *Advances in Cryptology (CRYPTO’99)*, pp. 388–397, 1999.
5. R. Anderson, M. Kuhn, “Tamper Resistance—A Cautionary Note,” *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pp. 1–11, 1996.
6. G. Suh, S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation,” *DAC 2007*, pp. 9–14.
7. M. Potkonjak et al., “Hardware Security: From Devices to Systems,” *IEEE Design & Test of Computers*, vol. 32, no. 2, pp. 64–75, 2015.
8. S. K. Saha et al., “Intrusion Detection Systems in Embedded Networks,” *Embedded Systems Journal*, vol. 7, no. 1, pp. 21–33, 2018.
9. Karri et al., “Trustworthy Hardware for Secure Embedded Systems,” *IEEE Security & Privacy*, vol. 10, pp. 54–62, 2012.
10. M. Mahmoud et al., “AI and Machine Learning for Embedded System Security,” *IEEE Access*, vol. 8, pp. 12345–12360, 2020.