Considerations for Developing a Secure Embedded Product

by Anil Janardhanan | January 03, 2024

We are in an era dominated by interconnected technologies, where demand for embedded products is increasing. From smart home devices to industrial automation systems to medical devices, the reliance on embedded systems has become ubiquitous. However, as our dependency on these products grows, so does the need for robust security measures. Welcome to our blog, where we embark on a journey into the realm of developing secure embedded products. Let's explore the challenges, best practices, and innovative approaches that engineers and developers must consider while creating embedded systems for today's interconnected world.

In the USA, recent National Cybersecurity Strategy proposes to shift the liability for insecure products and services from the consumer to the manufacturers or providers. Europe, too, will soon release regulations to very similar effects. The importance to consider security at all relevant embedded devices is much clearer now. Yet, security has been neglected for too long for embedded systems.

What is a secure embedded system?

Security in embedded systems is the effort to protect it from malicious access and use. Embedded systems are increasingly becoming targets of cyber-attacks. Examples are plenty - A smart refrigerator that allowed hackers to steal owners' Google credentials; a temperature sensor in a fish tank at a casino exposing a database of "high rollers" etc.

Securing an embedded system could involve a diverse array of strategies - encompassing encryption, secure boot processes, access control mechanisms, secure communication protocols, and intrusion detection systems. These measures are specifically crafted to fortify the system against an array of potential threats, including man-in-the-middle attacks, denial of service (DoS) attacks, and other forms of exploitation.

The requisite level of security for an embedded system depends on its intended application and the potential impacts of a security breach. A medical device responsible for managing medication delivery might necessitate a more elevated security standard compared to a basic temperature sensor embedded in a home thermostat.

We need to be aware that security is never perfect - a balanced and continuous approach to assess threats, risks, and mitigation is needed throughout the life of a product. Also, security features might increase systems cost – analysis, selection, and implementation of best practices to ensure optimal security for a product takes time, effort, and cost. Incorporation of specific hardware features, enhanced memory etc. essential for security will increase the overall system cost.

However, security should be an essential consideration in the design, development, and deployment of embedded systems to maintain safety and integrity of the system and its users.

Secure Software Development Lifecycle

A best practice recommended is to define and follow a Secure Software Development Lifecycle with security activities incorporated at each stage of development. This includes identifying potential security risks & vulnerabilities early in the development process, define and implement security requirements, and integrate security testing and validation into the lifecycle stages. By following a well-designed secure development lifecycle, we can reduce security vulnerabilities and improve the overall security and quality of the software – preventing data breaches, protecting user privacy, and ensuring system reliability & trustworthiness.

Following are some critical considerations for creating a secure embedded system:

1. **Threat Modelling:** Threat modelling is the process of identifying potential threats, determining the likelihood and potential impact, and then identify countermeasures to mitigate. Begin by identifying potential threats and vulnerabilities specific to the system

under development followed by a thorough analysis to understand potential attack vectors and its impact on the system.

2. **Secure Boot and Firmware Integrity:** Implement secure boot mechanisms to ensure that only authenticated and unmodified firmware is loaded during the boot process to prevent unauthorized or malicious code from running on the device.
3. **Authentication and Authorization:** Implement tiered authentication mechanisms to verify the identity of users, devices, and components. Additionally, enforce proper authorization policies to control access privileges and limit exposure to potential attacks.
4. **Review the APIs and functions**: During the detailed design and implementation stage review functions and APIs used and prohibit any functions that are determined to be unsafe for the application.
5. **Perform thorough static and dynamic analysis and Testing**: Perform static analysis, unit testing, and integration testing followed by run-time analysis & testing. Use run-time analyser tools that enables monitoring of CPU and memory usage leading to the identification of vulnerabilities related to memory leaks, buffer overflows, and stack overflows.
6. **Data Encryption:** Employ encryption algorithms to protect sensitive data both at rest and during transmission. This will include encryption of data on the devices and implementation of secure communication channels to prevent eavesdropping and tampering.
7. **Update Mechanisms:** Establish a secure and reliable method for updating the firmware and software. Regular patch updates to address emerging threats and ensure the long-term security of the device is recommended.
8. **Physical Security:** Consider the physical aspects of security, including tamper-resistant mechanical designs to protect against physical attacks.
9. **Network Security:** Implement network security measures to safeguard communication between embedded devices and other systems. This may include firewalls, intrusion detection systems, and secure protocols to prevent unauthorized access and data breaches.
10. **Minimize Attack Surface:** Attack surface reduction aims to limit the ways in which an attacker can gain access to or compromise the system. We can reduce the system's attack surface by disabling unused hardware and firmware features, using secure boot, limiting permissions, and following the principle of least privilege by limiting the access to only the resources they need to operate effectively and securely.
11. **Secure Debugging:** Ensure debugging interfaces and mechanisms are securely implemented and disabled in production environments. Unauthorized access to debugging interfaces can expose critical information and compromise system security.

12. **Continuous Monitoring and Logging:** Implement monitoring and logging mechanisms to detect and respond to security incidents in real-time. Regularly review logs to identify anomalous behaviour and potential security threats.
13. **Compliance to Standards:** Adhere to relevant security standards and industry best practices, such as ISO/IEC 27001.

In today's interconnected landscape, enhanced security of embedded systems is essential to ensure resilience and trustworthiness.