Building security into IoT devices by software isolation

By Yann Loisel, Stéphane Di Vito, and Emmanuel Puerto

The internet of things (IoT) is not only a buzzword, it’s a reality: it encompasses scenarios and situations that all share the same security requirements around communication, assets, and integration. In an IoT device, the communications must flow securely, the assets must be securely stored and processed, and the integration of the device must be consistent from a security standpoint. In this article, we will examine how you can build security into your IoT devices.

Let’s first discuss all the components of an IoT device: the architecture, communications, integration, and assets.

**Architecture**

Building connected objects means combining many functions, such as the ability to perform multiple processing tasks, to secure some data, and to communicate with back-end entities such as gateways, servers, or the cloud. Nevertheless, this impressive set of functionalities must be integrated into the smallest, cheapest, and simplest embedded system design.

This aggregation of features leads to an even more important exposure to attackers. Attackers see the assets handled by the IoT devices as their goldmine and their communication interfaces as the gate to the assets, facilitated by a permissive integration and design.

**Communications**

An IoT device is not an island in the middle of the ocean; it’s a link in the chain from data acquisition to data exploitation, or between the command control and the operation. Such a device without any communication interface is useless: it would be impossible for it to exchange data, or update its firmware and services. This means that an IoT device must always be, wired or wirelessly, linked to a network.

**Integration**

Once the hardware device is in the field, we can say the main milestone has been reached: it means that an important enough business case has been identified, justifying the development, the manufacturing, and the deployment. That’s great, but the next objective is now to exploit this device for the longest lifetime. The best way is to improve and add features, and to add applications. The consequence is that the same platform and the same microcontroller shall host several different applications. These applications have then to coexist, concurrently executing to address the device features.

In common knowledge, talking about multiple applications implicitly leads to a rich OS, with memory isolation mechanisms. The consequence is that only high-end CPUs (e.g. based on an Arm Cortex-A core) with large memories (RAM, flash) achieve these requirements. This appears to be a strong contradiction with the primary goals: “smallest, cheapest, simplest.”

**Assets**

This device handles assets, including sensitive and valuable data. Whether it acquires, modifies, sends, or stores the data, in the end it exposes this information. This security requirement also adds some complexity.

**Case study**

Let’s consider a representative example: an embedded device attached to a computer, able to read and handle fingerprints, to communicate in bidirectional mode using a wireless channel (Bluetooth, Li-fi), to host the fingerprint minutiae verification information and a secret key, and to run applications. This device’s first purpose is to control users’ access to a highly valuable service located on a remote server, accessible via a client computer.

This is the two-factor authentication method. When access to the service is requested by a user, a random number (a challenge) is generated and sent by the server to the embedded device via the computer. Then, if and only if the owner fingerprint matches the stored minutiae, the challenge is cryptographically linked (encrypted or signed) with the secret key, and finally sent back to the remote server for authentication.

To increase the lifetime of this device, new applications can be added to provide additional access controls to other high-value services. The biometric capability is very attractive and can be the start for many innovative services, like payment.

Yann Loisel, Stéphane Di Vito and Emmanuel Puerto work at Maxim Integrated as Security Architect, Security & Software Expert and Embedded Software Expert, respectively, in the company’s Micros, Security and Software Business Unit - www.maximintegrated.com
This kind of device is generally constrained by the price and the form factor. Therefore, the choice of the microcontroller running the device leans in favor of the smallest embedded memories and the simplest architectures: the modern choice is the Arm Cortex-M processor. Whatever the software development model, either internal team or subcontractor, the complexity and the variety of the embedded code will increase tremendously when moving from single to multiple applications. Consequently, since the Cortex-M core only features a memory protection unit (MPU), there is a very high risk that the device’s software will adopt a “flat model” where all applications share the same privileges and resources. In this model, only good practices, the good will of application developers, thorough code verification, and ... luck can guarantee that each application will not “disturb” or pollute others.

**Problem**

The highest risk of a flat architecture model is that any weakness, any failure in any application can impact all the other applications, hence the functioning of the whole device. As there is no real protection and segregation, an attack can spread from the weakest application to the others because of the uncontrolled resource sharing and the lack of execution privilege distinction.

Furthermore, the preferred attack channel is the external communication interface, which is an easy path for an attacker. It is, by definition, accessible from outside the device, ideally remotely accessible over a network. The software stack handling the communication is always functionally validated, but very rarely verified security-wise.

An attack path targeting a weakness in a communication stack will typically leverage a buffer overflow or a lack of input data check: an attacker may inject malicious code in the microcontroller’s internal RAM via the improperly checked communication buffers. Once this code is within the platform and executed, the attacker can take full control of the software running on the device. The consequences can be various, only limited by the attackers’ imagination: stealing biometric information, performing relay attacks, and any other scenario based on the embedded applications.

**Solution**

To counter that threat, software solutions leveraging hardware security mechanisms such as a memory management unit (MMU) or Arm TrustZone are usually proposed. They are “isolation” software, named hypervisors or “virtual machines.” They use hardware mechanisms to define separate software “boxes” or “containers,” each containing an application. The hypervisor’s main objective is to allocate resources to each of these boxes, to control these resources’ usage, and to control the communications between these boxes. It is commonly thought...
that such an approach is impossible on Arm Cortex-M-based microcontrollers because they do not have an MMU, rather only an MPU.

The DeepCover Security Framework (DSF) software isolation solution proposed by Maxim proves that such a high level of security can be reached, and guarantees a very strong isolation between the software “boxes” running on Cortex-M-based microcontrollers. The core of this solution relies on a hypervisor that enforces a strict software architecture configuration at any time. This configuration defines boxes, the boxes’ resources and the boxes’ interactions. Different applications run in separate boxes, each with their defined set of privileges and resources. Applications may interact with each other through gateways. Everything described in the configuration is guaranteed by the hypervisor that guarantees that the boxes’ execution matches the configuration. Any difference with the intended configuration is immediately detected, such as access out of allowed memory ranges, attempts to interact with another box out of an allowed gateway, or direct access to disallowed resources. The hypervisor manages and controls other boxes’ resources and execution, while keeping its own ones for its execution.

This solution is credited with a high level of confidence. Indeed, a security laboratory has rated the framework as complying with the very demanding PCI-PTS POI security requirements for software isolation: what is secure for the payment can be considered secure for the IoT too.

In our example, we see that DSF allows a simple but efficient architecture design that consists of the exclusive allocation of a specific box for the management of the biometric characteristics, including access to the fingerprint sensor and the extraction of the minutiae. By preventing access to this sensor from other boxes, and thanks to the hypervisor’s control over this access, any fraudulent or unwanted attempt to read the sensor is detected and an exception is raised, leading to a safe failure. Following the same model, other boxes can be defined and developed with a focus on their very specific functional objective, after a careful study of their needs in term of resources and peripherals. The communication links between boxes are enforced by the hypervisor, which guarantees that only these links can be used for data exchanges between the boxes. No arbitrary jump to a random code location is possible, and no direct access to another’s box resources is possible.

Of course, this framework (or any other, even with any hardware support like the MMU or TrustZone) neither improves developers’ skills nor transforms any buggy software into a bug-free application, but yet the attack scenario described above is mitigated: a weakness in an application cannot expose the whole firmware. The bug remains contained in its box, hence the bug’s value is dramatically decreased.

The immense benefit of this solution is to allow hosting of multiple applications from multiple providers with different levels of confidence and robustness on a single device. The owner of the most demanding application can now accept other applications on the same hardware device without the risk of weakening his own application and exposing his assets.

Benefits

By extension, the high level of control in the boxes’ architecture and resource partitioning allows the update of applications easily and with trust. Applications are fully controlled by the hypervisor, which can modify, update, or remove them through simple operations without hurting the functioning of the others. Furthermore, that secure update can propose additional services like encryption without exposing the keys to any box.

Finally, another important benefit is that, as one box execution does not hamper another box execution, any certification granted to one box is not impacted by another box modification: the certification remains valid. From a single monolithic software, where each modification has an impact on its whole robustness and the certification of some portions of it, we have now moved to a set of independent, isolated images, keeping their own properties and bugs!