Final Design Document: Secured Architecture for Medical Devices

Team Maverick

April 1, 2024

1 Introduction

In this final design document, we outline the challenges and insights we inferred while implementing our proposed design, a robust and secure architecture for a set of micro-controllers, i.e., `MAX78000FTHR`. We configured one of the controllers to be our application processor (AP) and the rest to be the associated components (c0, c1,...), designed to function as a complete medical device with its own auxiliary components.

The primary goal have been ensuring the integrity and confidentiality of sensitive data stored within these devices while safeguarding against potential attacks from adversaries. The host tools or the `ectf_tools` as defined in the reference design repository, would ensures a consistent interface and we would be designing the architecture satisfying the security requirements. For this, we mainly focused on the following tasks:

- PIN and token validation
- Securing post-boot communication

2 System Overview

The system architecture is mostly similar to reference design and mainly comprises of two main components: the **AP** containing the **pin** and **token** values and the **components** containing the **attestation data** such as location, date, and customer name. The flow of the architecture as given in the reference design was simple and easy to understand. A set of four functions that were called from the main, list(), attest(), replace(), and boot().

The functions in turn, activated many other components which did the communication in the background such as the attest() activated attempt_attest() function which at first, validated the pin and after successful verification, called the function attest_component(). The attest_component() sent a packet to the component based on the component_id and the corresponding opcode, in this case it was "COMPONENT_CMD_ATTEST" which was recognized by the component side to relevant operation (here, process_attest()). Each of the function went through similar workflows and gave the result accordingly.

2.1 Security Components

Initially we proposed to implement the ECC encryption for security the medical device. However, implementing the same even with libraries turned out to be very challenging. Furthermore, we observed that the pin and token values were being compared on the AP end with the buffer input provided in the CLI.

Therefore, to our understanding, hashing the PIN after it has been initialized by the ectf_tool and kept in the *ectf_params.h* file when we received the buffer input did not secure the overall design. Hence, given interest of time, we ensured that the values and length of the AP_PIN and AP_TOKEN did not exceed the maximum size allocated to them. We also made sure that these values are "null terminated" to avoid buffer overflows. We also added time delays if there is an attempt to guess the pin and token values by an adversary.

After validating the components and AP using a simple validation function, we initiate the boot() after the device goes into POST_BOOT process. Initially our understanding was to create our POST_BOOT functionality on the basis of AES symmetric encryption. We tried to solve the challenges around getting the wrapper work. However, the testing server kept giving time outs. We came to a conclusion that we might need a lightweight security component that was easier to calculate and does not require to be immensely complicated since validation of the component has already been done. We utilized a strong hashing algorithm SHA256 on some of the portion of the buffer and concatenated the hash value with the buffer. On the receiver end the hash value is matched with the claculated hash value of the same portion in the buffer. It is a simplistic yet an effective way of securing communication after establishing that all the components are verified.

2.2 Example Scenario

Consider a scenario where AP needs to retrieve attestation data from one of the Component:

1. A user needs the attestation data in the component and uses ectf_attest and provides the pin and the component id.

2. The AP first validates the pin and then sends a packet to the component with corresponding instruction to retrieve the attestation data from the component.

3. The Component receives the packet and checks for the operation code and based on the instruction provided by the AP, it sends the required information.

3 Conclusion

The designing phase made us learn how to think about a secured architectures. However, implementing the same in physical devices with properly working routines is a non-trivial task. By implementing a simple validation process we ensure the pins and token get validated, and we also incorporate delays to minimize brute force attacks. We establish secure communication channels for data exchange through the application of SHA256 hashing algorithm. This design prioritizes both security and efficiency, which are essential for the integrity of medical systems in today's digital landscape. Although libraries like WolfSSL help immensely in minimizing the barrier for beginners by providing pre-defined functions on state-of-the-art algorithms for security. However, implementing or fast prototyping on the same might not be trivial.