2024 eCTF: Submission Document

Team IITI

Introduction

This document outlines the design of the solution implemented by our team for the Embedded Capture The Flag (eCTF) competition hosted by MITRE. The solution aims to address the challenges presented in the competition, focusing on secure and efficient strategies for securing the Medical Device. We have given a simple overview of our implementation of the security requirements.

Security Requirement 1

The Application Processor (AP) should only boot if all expected Components are present and valid.
The following problem requires us to deal with two main parameters:

1. All the expected components are present.
2. We must verify all the components are valid.

This is important to make sure that all the components are valid and expected to ensure unintended actions by the Medical Device. To accomplish this, The global secrets contain one key: A key for the components. The extra XOR operation between the ComponentID and the component key in the authentication process adds an additional layer of security to the verification mechanism.

Whenever the AP wants to verify the component, it calculates the XOR of the component key and the provisioned ID, which is used to encrypt a puzzle where we add a random number generated by the **TRNG** and the provisioned ID. The component receives the message and first decrypts the message using the XOR of the ComponentID and key. It should only be able to decrypt if the ComponentID and provisioned ID match. Even then, as an extra layer, we subtract the ComponentID from the decrypted message and send it back to the AP. Here, AP compares the returned message, the random number and the original random number. If they are found to be equal, the component is validated. We perform the above process for all the provisioned devices. Only then can the boot function be called from the AP.

Security Requirement 2

Components should only boot after being commanded to by a valid AP that has confirmed the integrity of the device.
This requirement should be satisfied after confirming the integrity of the device by integrity; we assume that it refers to security requirement 1. We plan to use the random number used in the puzzle communication. The same random number will be hashed using **MD5** stored for each component in a map in the AP, with keys as the provisioned ComponentID and the value as the hashed random number. The component will also store the hashed random number that was sent to it during the validation process.

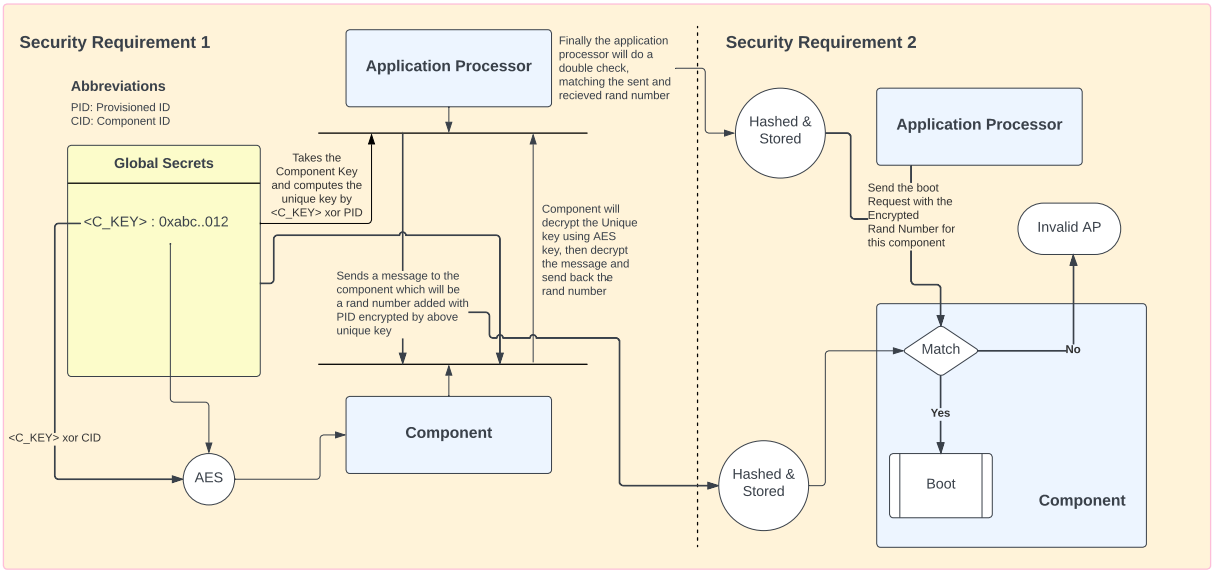

Figure 1: Security Requirement 1 and Security Requirement 2 Flow Chart/s

While booting the components, we will send the hashed random number for a particular component while sending it the boot command; the component will first check it against the one stored in it. If it matches, it will call the boot sequence; otherwise, it will return an error message.

By requiring components to match the hashed random number during the boot process, the system ensures that only components are commanded by a correct and validated AP. Intertwining the steps for Security Requirement 1 and Security Requirement 2 creates a cohesive security framework, fortifying the system against potential threats and unauthorized access throughout the lifecycle of the components.

Security Requirement 3

The Attestation PIN and Replacement Token should be kept confidential.
The problem requires us to deal with the issue that if we store the Attestation PIN and Replacement Token directly on the flash memory of the AP, then it will be susceptible to attacking and getting the Attestation PIN and Replacement Token.

So, to keep the Attestation PIN and Replacement Token confidential while building the AP, we first encrypt the Attestation PIN and Replacement Token using **SHA-256**, a one-way or irreversible hash function. While requesting the Attestion and Replacement of components, we will first encrypt the Attestation PIN and Replacement Token using the same one-way function and compare it with the stored hashed versions of the tokens.

Security Requirement 4

Component Attestation Data should be kept confidential.

For this, we will keep attestation data in the component in encrypted form using **AES encryption** with the key as XOR of C_key and ComponentID to ensure the key is unique to each component.

Firstly, the user requests the attestation data by entering the PIN. The PIN will be stored as a hash, which will be compared with the PIN entered by the user after hashing it. If a match is found, the AP sends a request to the component for the required data. The data will be sent to the AP using the secure send and secure receive functionality. As AP has access to C_key and ComponentID, it can decrypt the received message and display it.

Security Requirement 5

The integrity and authenticity of messages sent and received using the post-boot MISC secure communications functionality should be ensured.
To satisfy the given requirements, we will use **AES encryption/decryption** to secure the channel. Whenever any component wants to send a message to AP, it will encrypt the message with C_KEY XOR ComponentID. When the AP receives the message, it will decrypt using the same KEY, i.e., C_KEY XOR ComponentID, and vice-versa. Furthermore, using this way of creating encryption keys will also benefit us while replacing the components, as the AES encryption key used in generating the actual encryption key will remain constant but the actual encryption key would be distinct and change as per the component. This way, it also aligns with the assumption that the global secrets file will not change after deployment, but still the encryption keys stay dynamic.

Implementaton Details

Build Deployment

We randomly generate a C_KEY of 16 bytes using a Python script that is used in various AES-based encryption and decryption communication.

Build Setup

We add an initial setup stage in building AP and Components:

1. **AP:** We hash the provided Attestation PIN and Replacement Token using **SHA-256** before building the elf file.
2. **AP:** We encrypt the provided Attestation Data using **AES**, C_KEY, and Component ID as the AES key before building the elf file.

This is done using a Python script (Utils/script.py).

Timeouts to Prevent Brute Force Attacks

We add timeouts in the case of wrong Attestation PIN and wrong Replacement Token.

Message Format

Most of the messages in our design are taken directly from the ECTF Example Design. There are a few changes

1. **Attestation Data:** The encrypted attestation data from the components are sent to AP, in this format: The first 80 bytes correspond to the encrypted Attestation LOC, the next 80 bytes encrypted Attestation DATE, the next 80 bytes encrypted Attestation CUST, the next 3 bytes store the actual message length of LOC, DATE, and CUST.

2. **Post Boot Messages:** The encrypted messages from the AP/Component are sent to Component/AP in this format: The first byte contains the actual message len, the second byte contains the padded message len (to the nearest ceil multiple of 16), the third byte contains the length of the encrypted message. The next length of the encrypted message bytes contains the actual encrypted message.

3. **Validate Message:** We added a hashed random number field of 16 bytes that is used to get the random number from the component.

4. **Command Message during Boot:** We pass the stored random number in the AP to the component for verification by passing the hashed random number in the params field.