UNITED STATES COAST GUARD ACADEMY
Department of Engineering (EE/CYS)
eCTF 2024

Design Document

Medical Infrastructure Supply Chain (MISC) System Security Initial Design Document

1/c Cole Fulton

1/c Lara Taha

1/c Orwin Austin

1/c Matthew Reheuser

1/c Ryan Vigo

I. Introduction

In regard to the functional requirements for the Medical Device due to this being the first year of competition for the United States coast Guard academy we will be leveraging the reference design produced by the organizers. Using this as a framework will accomplish and meet the functional requirements for us. We will then be coding in completely C to accomplish all security requirements that are missing from the reference design.

II. Security Requirements

Security Requirement 1:

The Application Processor (AP) should only boot if all components are present and valid.

This security requirement requires the Application Processor to only boot if all expected components are present and valid. To secure our design against this flag we will utilize a random number (nonce). To implement this, we defined a random number in the global secrets header file. For the Application processor (AP) to check the validity of the component, the component sends its component ID added with the nonce to the AP. The AP then subtracts the component ID which is known to the AP and compares the nonce received to the global secret stored nonce. This method will allow for the AP to use a simple function to easily check if each component is valid before booting.

Security Requirement 2:

Components should only boot after being commanded to by a valid AP.

The second Security Requirement states that the components should only boot after being commanded to by a valid AP that has confirmed the integrity of the device. This requirement is crucial in preventing unauthorized use of components, which helps safeguard patient safety, protect sensitive data, and maintain the company's reputation. Like SR1, we will use a nonce to validate the AP in order to boot the components. Similar to the process in SR1, the AP will send the nonce stored in the global secrets header file to the component. The component will then compare the nonce received from the AP to the nonce stored in the global secrets header file. If the two nonces match, then the AP is valid, and the component can boot. This method allows for the Components to verify that the AP is valid using a random number that can only be obtained from the global secrets header file.

Security Requirement 3:

The Attestation PIN and Replacement Token should be kept confidential.

Now focusing on the implementation of the 3rd Security Requirement, The Attestation PIN and Replacement Token should be kept confidential. This security requirement is used to protect the system from the “Operation Pin Extract” flag. For this flag the attacker is given access to a fully working medical system however does not have access to the PIN or the Replacement Token. This security requirement will be accomplished through encryption of the Attestation PIN and Replacement Token. For the encryption we will use a Python script to encrypt the pin and token in the makefile using pycryptodome’s encryption library specifically AES. The Python script will then append the eCTF parameters header file. We then will use the Simple Crypto Library to encrypt the user input to be compared to the stored pin and token. For the AES we will use a 128-bit key generated using a Python script. The AES key will be passed

to AP through the Global Secrets header file. While this approach is simple it should provide a simple deterrence to gaining the “Operation Pin Extract” flag for attacking teams.

Security Requirement 4:

Component Attestation Data should be kept confidential.

For Security Requirement four to be satisfied, we need to ensure that the Component attestation data is kept confidential. The attestation data on each Component is essential to determining the validity of that Component. If an attacker can access this data without the required privilege, they may be able to recreate or modify the critical Component. To meet the security requirement, we will be using Advanced Encryption Standard (AES) to encrypt the location, date, and customer data. Similar to SR3 we will be using a series of Python scripts to encrypt the information in the make file. This encryption will be using the same key as SR3. We then use simple crypto library to decrypt the attestation data after the AP pin was validated, and then print the data. This process allows for the attestation data to remain confidential until given a valid AP pin.

Security Requirement 5:

The integrity and authenticity of messages sent and received using the post-boot MISC secure communications functionality should be ensured.

The Secure_Send function, both on the AP and component, starts by taking in the buffer. This buffer is then padded with null characters to achieve a uniform size to maintain consistency and security across all messages. Following this, we utilized Simple Crypto library – AES

encryption and encrypted the padded buffer. After the encryption, the data packet is transmitted. Once it is received, the data packet is decrypted utilizing Simple Crypto library and the same AES algorithm. Finally, all null character padding is removed from the decrypted buffer to retrieve the original message content.