Oklahoma Christian University

Software Design Document v3.0

Embedded Capture the Flag Competition

Description

The software design of the eCTF project seeks to implement a representation of a medical device and solve the problem of an insecure medical device. The main subsystems of the software include the application processor, components, deployment, build environment, and host/build tools. These subsystems work together to allow the host tool to interface with the medical device consisting of the application processor and components. The subsystems will also implement functionality to prevent attackers from accessing sensitive information on the medical device.

Architecture

The architecture of the software consists of 5 main subsystems. The organizers of the eCTF competition wrote the code for these subsystems.

Application Processor

The Application Processor subsystem is the subsystem that interacts with the UI and commands the components via an I2C connection. The AP handles the scan, boot, attest, and replace commands from the user. The functions that handle these commands are defined in “application_processor.c” and are detailed below.

- void init()-function

    - Function initializes the AP on startup by enabling global interrupts and flashing data such as the component IDs onto the device.
- int issue_cmd(i2c_addr_t addr, uint8_t* transmit, uint8_t* receive)
    - This function sends a command from the AP to the specified component via the I2C connection.
- int scan_components()
    - This function prints out the provisioned component IDs of both devices. This function meets a functional requirement.
- int validate_components()
    - This function validates that the components connected to the AP have valid component IDs. In this function the AP commands the two components to send them their corresponding IDs and the AP compares the inputs to the ID it has stored in its flash. If the components are not valid the AP should not boot.
- int boot_components()
    - The AP issues the boot command to the components after the AP is booted.
- int attest_component(uint32_t component_id)
    - This function is called in the attempt_attest() function. If the user has a valid attestation PIN. This function will command the specified component to send its attestation data. Then the function will display the data on the UI.

- void boot()
    - This function runs to declare that the booting of the AP was successful. When this function is called the AP will start running its post-boot functionality.
- int validate_pin()
    - This function is called in attempt_attest(). This function prompts the user to enter an attestation PIN. It then compares the inputted PIN from the user to the actual PIN. If the PINs do not match it will return an error.
- int validate_token()
    - This function is called in attempt_replace(). This function prompts the user to enter a replacement token. It then compares the inputted token to the actual token. If the tokens do not match it will return an error.
- void attempt_boot()
    - This function attempts to boot the device after it receives the boot command from the host tools. This function calls validate_components() to verify the integrity of the devices before the AP boots. If both components aren’t valid the AP will not boot.
- void attempt_replace()
    - This function attempts to replace a device after it receives the replace component command from the host tools. This function calls validate_token() to verify that the user has a valid replacement token. If the token is valid the function will go through the routine of replacing the component.

- void attempt_attest()
    - This function attempts to get the attestation data from one of the components after it receives the attest command from the host tools. This function calls validate_pin() to verify that the user has a valid attestation PIN. If the PIN is valid the function will receive and display the attestation data from the specified component.
- int main()
    - This function is the main code that runs on the device when it is powered up. The device is initialized and then waits for one of the four commands from the user which are: scan, boot, attest, and validate.

Component

The component subsystem is mainly used for storing medical data, sending and receiving data from the application processor, and booting a component. These functions are implemented in “component.c” in the component directory and are detailed below.

- void secure_send(uint8_t* buffer, uint8_t len)
    - This function is used to send packets over the I2C bus to the application processor if the component has been booted.
- int secure_receive(uint8_t* buffer)
    - This function is used to receive packets over the I2C bus from the application processor if the component has been booted.
- void boot()

    - This function contains the boot sequence for the component which indicates a boot has occurred by flashing the LEDs on the component.
- void component_process_cmd()
    - This function works to process commands from the application processor. This function works by implementing a switch statement that checks for the type of command received and then calls the corresponding function to the command received. If the command is not recognized the program control is returned to the main function.
- void process_boot()
    - This function boots the component and responds with a boot message to the application processor.
- void process_scan()
    - This function completes a scan by sending the component ID to the application processor.
- void process_validate()
    - This function works to validate a component by sending a signed component ID to the application processor.
- void process_attest()
    - This function responds to the attest command given by the application processor by sending the attestation data of the component to the application processor.
- int main(void)
    - The “main” function enables global interrupts and initializes the I2C address linking the application processor and board together. Then the function waits for a

command from the application processor. If a command is received the component calls the function component_process_cmd() previously described.

Deployment

The deployment subsystem generates any global secrets needed for the device firmware. This is accomplished by configuring the MakeFile to run files that will generate global secrets. The MakeFile generates the header file named “global_secrets.h” which can be accessed in the application and component subsystems to further implement security. The sensitive material generated in deployment is used for the build process of both the application processor and the components.

Build Environment

The build environment used is the Nix environment. The Nix environment contains all the compilers, packages, and other dependencies needed to build the firmware for the medical device. The code for the build environment setup is contained in the “shell.nix” file which configures the specific packages and dependencies to be invoked when the Nix shell environment is entered. One notable package built in the Nix environment is poetry which is used to interface the host tools. In addition, Nix also installs Make, Python 3.9, gcc-arm-embedded, Cacert, Minicom, Analog Devices MSDK, and OpenOCD. The directory “custom_nix_packages” includes an additional package in the file “analog_open_ocd.nix” that is used to build the Analog Devices fork of OpenOCD used for on-chip debugging functionality.

Host/Build tools

The host tools allow the host computer to interface with the medical device through the command line. The build tools work to construct the firmware of the device and flash the firmware to the application processor. The commands the host tool can send to the medical device follow the functional requirements. These commands include: getting attestation data from the application processor, booting the application processor and components, listing the current components, and replacing a component ID on the application processor. As per the guidance of the eCTF organizers, nothing in this directory should be modified.

Security Design:

To implement the security requirements the following changes will be made:

- Encryption Key Generation
    - Two key pairs will be independently generated using the website https://cryptotools.net/rsagen. These keys will be formatted into RSA key structures in order to utilize the wolfssl RSA algorithm. The two key pairs will be written to ap_secrets.h and comp_secrets.h via the deployment directory's makefile. Each key pair will be composed of a private and a public key. One pair will be for the AP and the second for both of the components. The AP will know its own key pair as well as the public key of the component. The components will know their own key pair and the public key of the AP.
- Component and Application Processor Validation
    - To authenticate the validity of the components the key pairs will be flashed as previously described in Encryption Key Generation. Changes will be made to the function validate_components() in the file application_processor.c which is

located in the src folder of application_processor. The structure “command” contains the variable named “params”. This variable will be populated with a string message signed using the function wc_RsaSSL_Sign(). This function signs the message with the AP key. This message is sent to the component. The component then utilizes wc_RsaSSL_VerifyInLine() to decrypt and verify the command was sent by a valid AP.

    - In the function process_validate() within the file component.c which is located in the src folder of component. The structure “packet” contains the variable “component_id”. This variable will be populated with a string message that is signed using the function wc_RsaSSL_Sign(). This function signs the packet with the component key. This message is sent to the AP. The AP then utilizes wc_RsaSSL_VerifyInLine() to decrypt to verify the packet was sent by a valid component.
    - After verification of AP and component validity, both are booted and communication is opened.

- Post Boot
    - The pre-boot functionality should be limited before the device has confirmed its integrity. Once the device and components, they should enter post-boot functionality. In post-boot functionality, the AP should communicate with the components via secure send and receive functions. These functions will utilize the wc_RsaPublicEncrypt() and wc_RsaPrivateDecrypt() functions to secure the messages transmitted.
- Confidentiality of attestation PIN and replacement token

- This is accomplished by using the WolfSSL library using the hash library. The specific hash used is the SHA224. The goal of the hash is to prevent a hacker from obtaining meaningful data if the device is intercepted.

  The method that defines the hash function is in the file named “simple_crypto.c” in the application processor directory within the src folder. The validate_token() and validate_pin() functions described below are in “application_processor.c” which is in the same directory as the previous file.
- The code operates in the following way:
  - The variables that hold the hashed version of the AP PIN and AP Token are declared in the global data space so that all functions can access these hashed values.
  - The original Attestation PIN and Replacement Tokens are hashed in the AP init() then hashes are stored in memory.
  - Within the validate_pin() function, the pin entered by a user of the device is hashed using the same hash function previously described. The hashed version of the entered pin is then compared to the hashed version of the correct pin. If the hashes match, the PIN is accepted. If they do not match, the pin is not accepted and an error message is displayed indicating to the user they have entered an invalid pin. This function flow diagram can be seen in Figure 4.
  - The process within the validate_token() function is identical to the validate_pin() process previously described with the exception of the AP

token being the point of comparison. This function flow diagram can be seen in Figure 5.

Diagrams

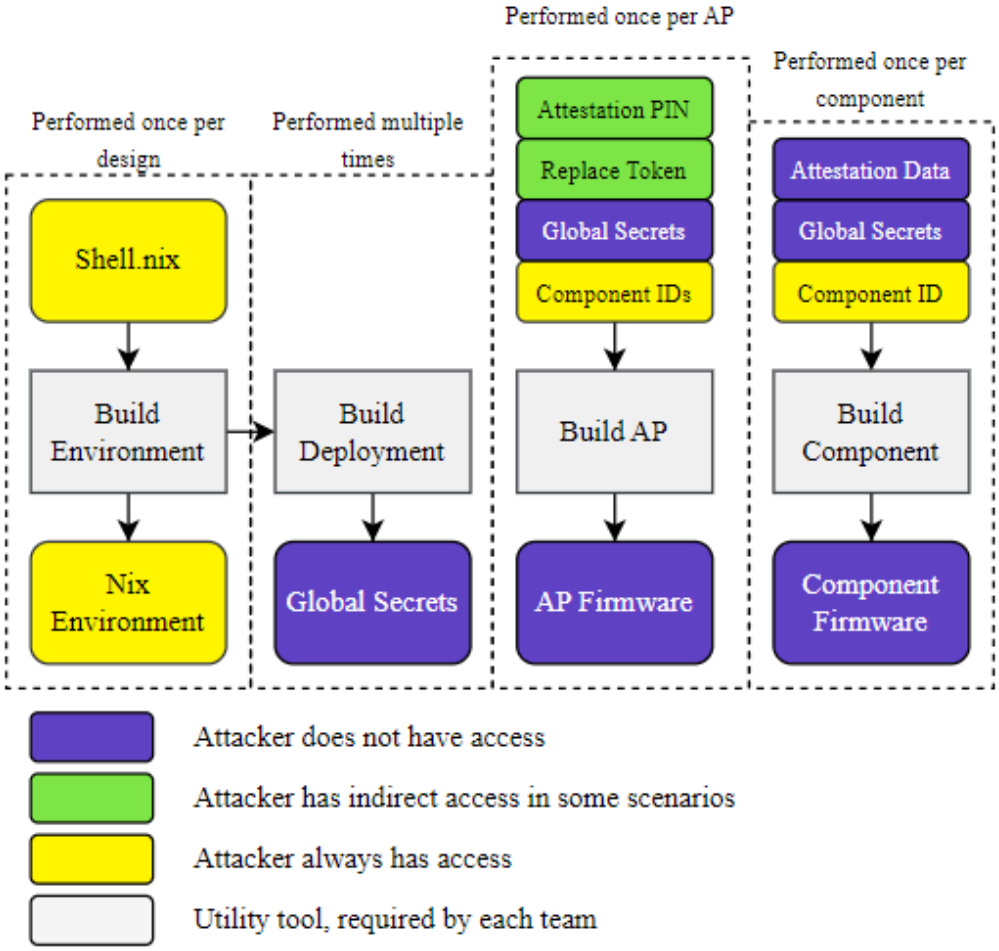

Figure 1 The build process diagram for the medical device provided by eCTF

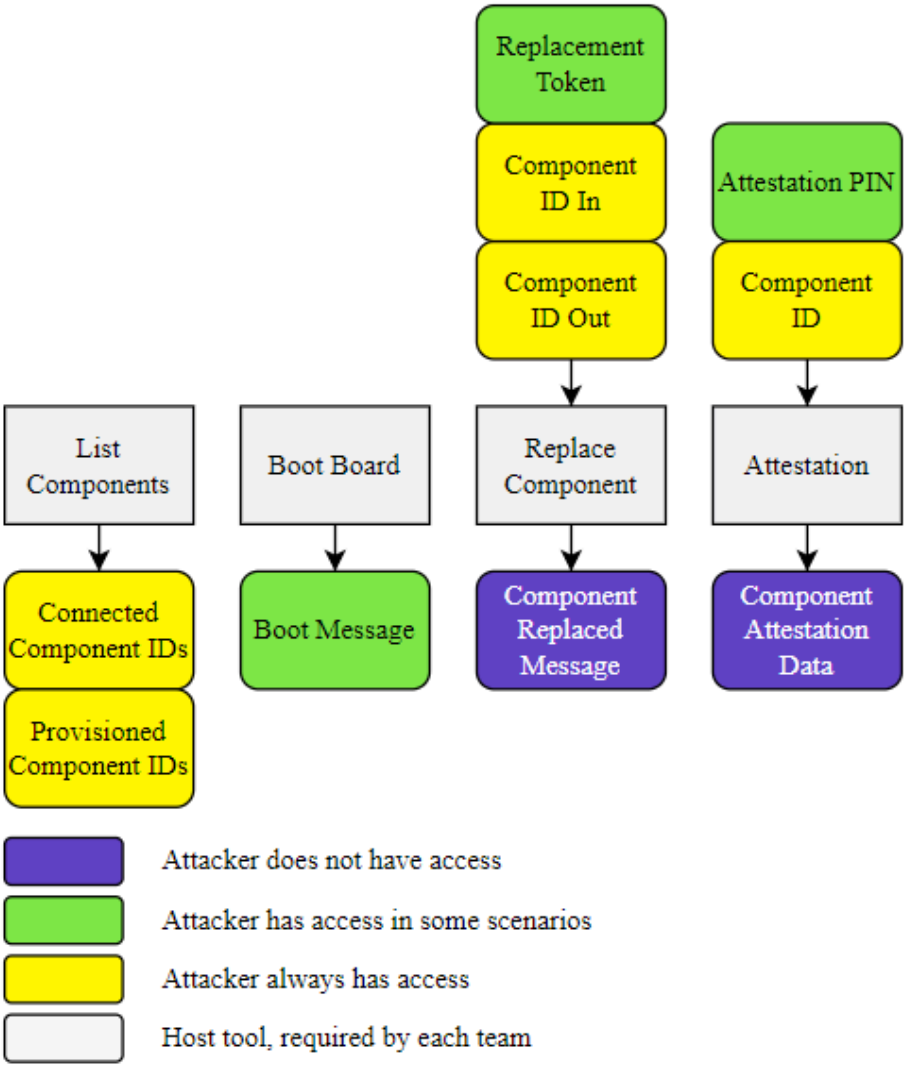

Figure 2 The Command Flow diagram for the medical device provided by eCTF

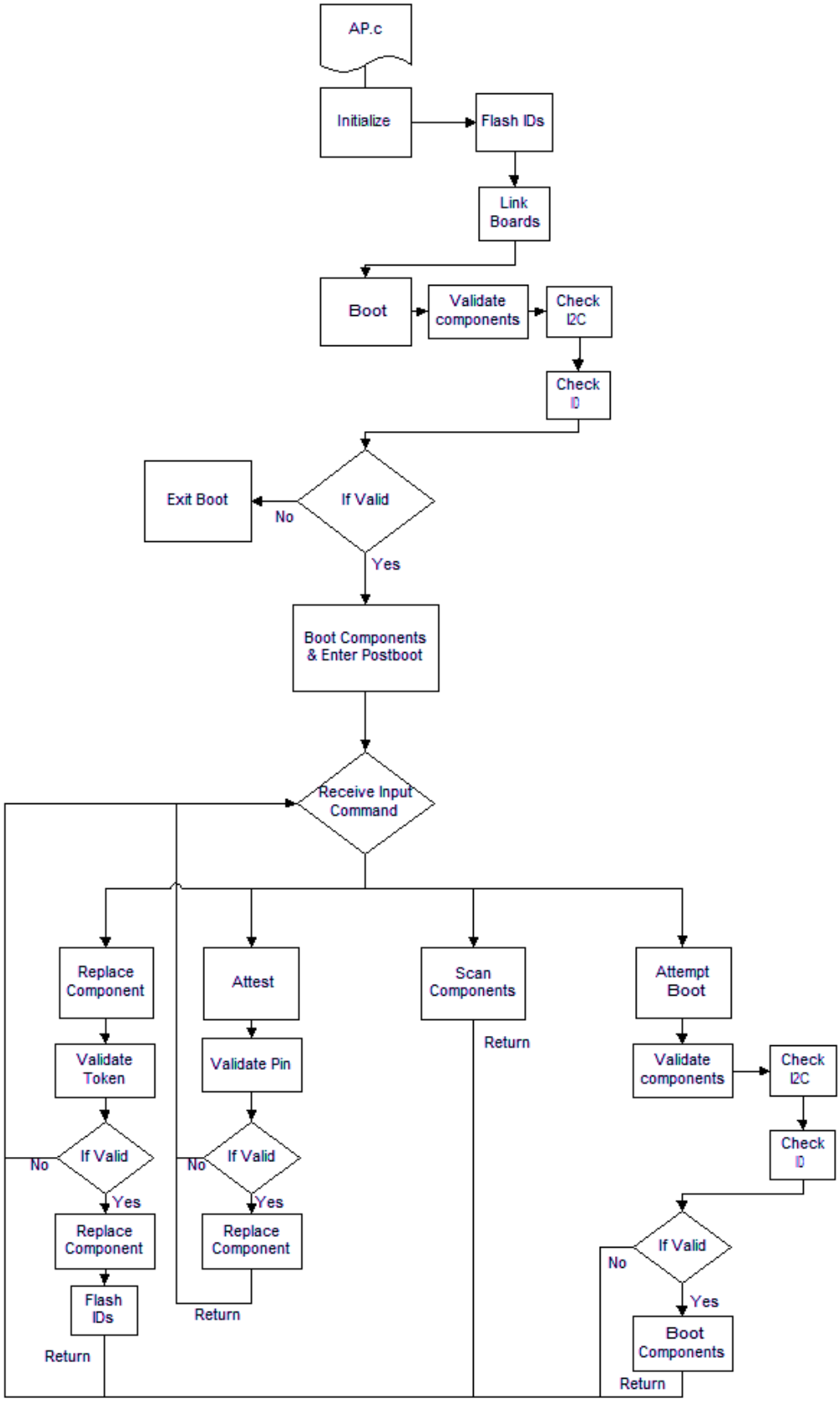

Figure 3 The Program Flow diagram for application_processor.

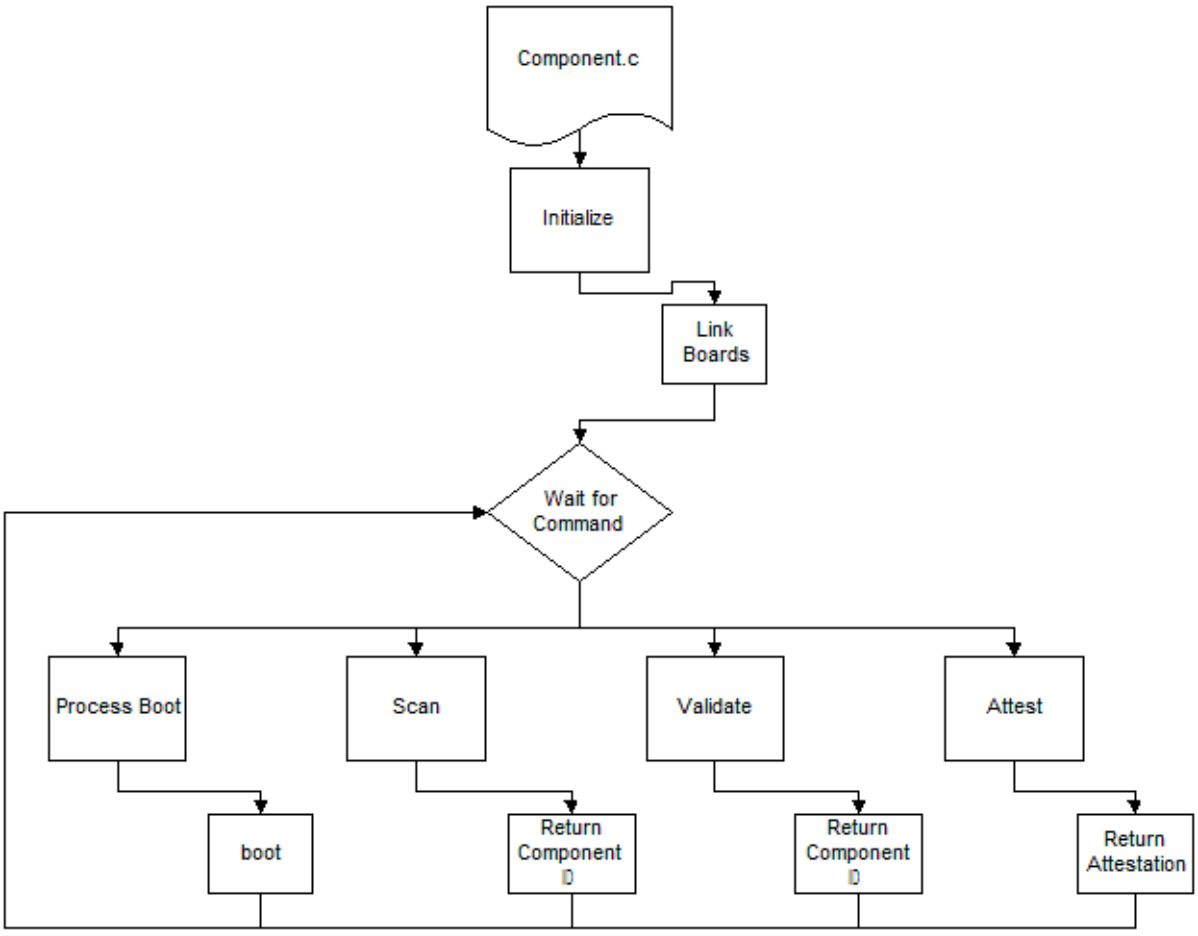

Figure 4 The Program Flow Diagram for component.c.

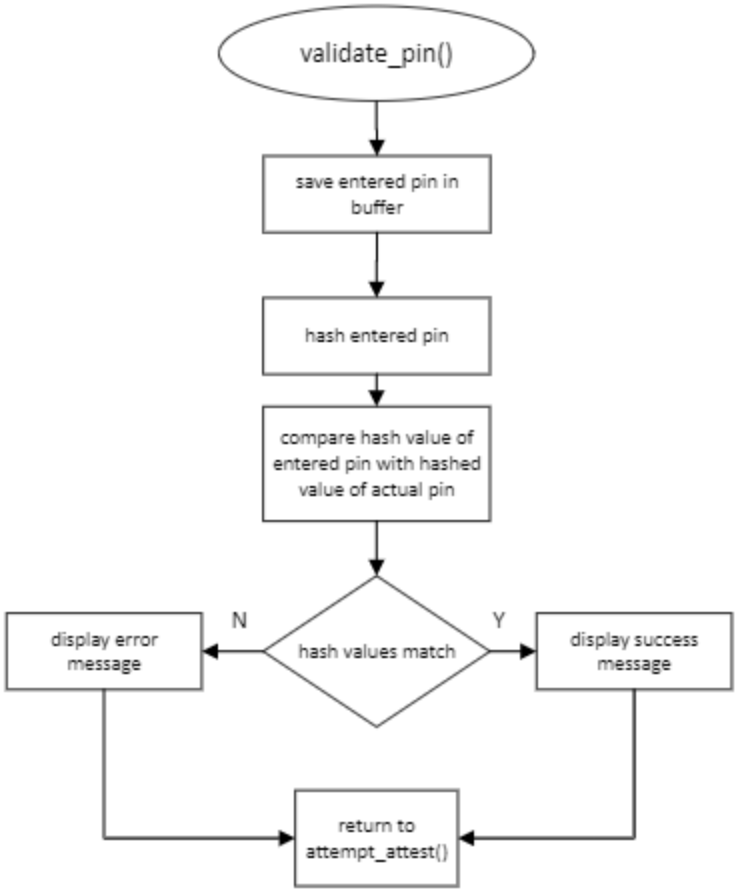

Figure 4 The function flow diagram for validate_pin() in application_processor.c.

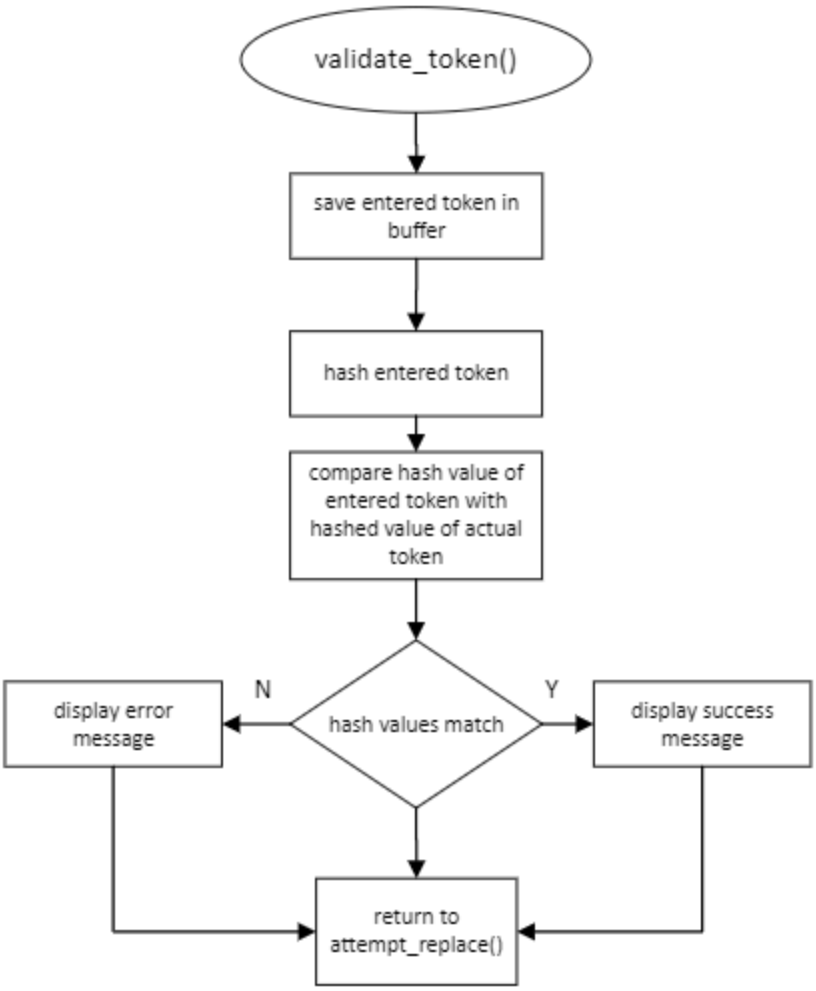

Figure 5 The function flow diagram for validate_token() in application_processor.c.