Medical Infrastructure Supply Chain (MISC)

System Design Document

University of Connecticut

K.L.W. Marquis & co.

eCTF 2024

Introduction

This document provides a detailed overview of UConn's 2024 eCTF design. This document is broken down into several sections. System overview describes the Medical Infrastructure Supply Chain (MISC) system at a high level. The Functional Requirements section describes the required functionality of the system as well as how the system fulfills these requirements. Threat Modeling describes the various attacks the system might be vulnerable to (if left unprotected) along with any solutions that are part of the design. Security Requirements details the various requirements the system has for security and privacy as well as how they are fulfilled. Stretch goals details a few ambitious goals for the system that are not required for bare-minimum security and functionality.

System Overview

The MISC system is composed of two parts: components and an application processor (AP). An application processor is essentially a master device, which controls the two components. The AP receives commands from a host computer that the user interacts directly with. The user does not interact directly with component devices. Instead, they must be accessed through the AP, which can release attestation information and perform other functions on valid, paired components. Security is provided through the custom protocol TLS-Lite, a pared down version of TLS-1.3 (see section entitled TLS-Lite below).

Functional Requirements

System Build Process

Build Environment

The Build Environment step remains unchanged from the reference design. The default nix-shell includes GNU Make, Python 3, an ARM cross-compiler, Poetry, OpenOCD, along with the `cacert` and `minicom` packages.

Build Deployment

The Build Deployment step primarily entails the generation of Global Secrets. During this step, the system generates a 1024b RSA key-pair for the deployment. This key fulfills the role of the root certificate for all devices in the deployment as the deployment itself is treated as a certificate authority (CA). This key pair is included in the Global Secrets file and is used to sign device keys generated in future device build steps. The public key is also included in all device binaries to verify other device keys.

Build Application Processors and Components

The Build Application Processor and Components steps work largely the same as one another. During each build step, a 32B ECC key-pair is generated for the device. The public key is signed by the deployment's private RSA key. Then, each device is provided with the following build-secrets: the device's signed ECC public key (which shall serve as its certificate), the device's ECC private key, and the deployment's RSA public key. The build configuration for components will include some attestation data that is stored on the device. Both Application Processors and Components include a boot message, which is printed upon device boot.

System Functions

List Components

The list components tool communicates with the application processor (AP) and lists the component IDs associated with all components currently paired with it. There is no private information associated with the list step, and thus the security concerns for this function are minimal.

The host computer sends the command 'list' over UART to the AP. Then, the AP responds with the component id's, which are stored in flash memory. Next, the AP scans each of the connected components, sending each of them a scan-command message. Each component responds with its component ID. The AP sends the connected component IDs to the host computer over UART, thereby ending the list protocol.

Attest

The attestation tool communicates with the application processor (AP) and lists the attestation data for the targeted valid component that is connected with the AP. This protocol applies TLS-Lite to authenticate the component and transmit the attestation data while maintaining secrecy.

The attestation tool sends the command "attest" over UART to the AP, along with a PIN and component ID. First, the AP checks whether the PIN is valid via a side-channel resilient comparison function. This function is implemented with the Double HMAC Verification protocol– a randomly generated key is used to generate an HMAC of both the PIN and the user-input, allowing the MACs to be compared instead of the raw data (see section "Double HMAC Verification" below). This prevents an adversary from obtaining sensitive information about the PIN from the timing side-channel.

If the PIN is invalid, the attestation command is aborted. Otherwise, the AP initiates a TLS-Lite-Handshake with the targeted component. This serves to verify the component is legitimate. Finally, after the handshake is complete, the AP requests the attestation data from the component. The component sends this data over the secure TLS-Lite channel (AES-128), using the symmetric key generated by the handshake. Only the legitimate AP that participated in the handshake is able to decrypt the data.

The high level attestation protocol is shown in Figure 1 below.

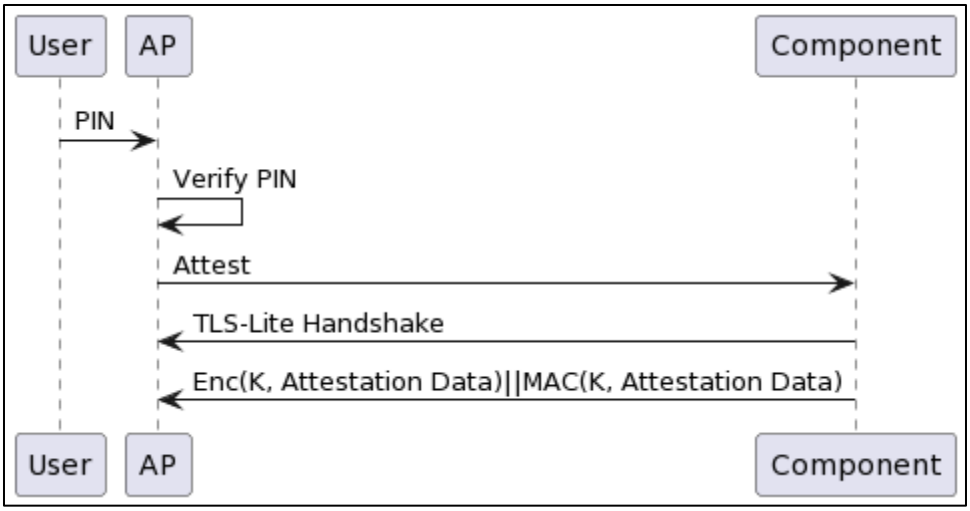

Figure 1. High level attestation protocol.

Replace

The component replacement tool communicates with the application processor (AP) and attempts to remove one of the existing, paired components, and pair a new valid component. After receiving the command, along with a replacement token input, the AP first checks whether the user's token is valid. This is done by comparing the replacement token with the correct token stored in on-board flash memory. A side-channel resilient comparison function that applies the Double HMAC Verification protocol is leveraged in order to prevent leakage of secret information over the timing side-channel. If the token is valid, the AP will replace the old (to-be-removed) component ID with the newly provided one. Note that it is not necessary to perform any validation of the new component at this step since such checks will be performed at the time of boot/attest commands. The fact that the user has access to the replacement token is sufficient to verify they have permission to alter the paired components. The high level protocol is shown in Figure 2 below.

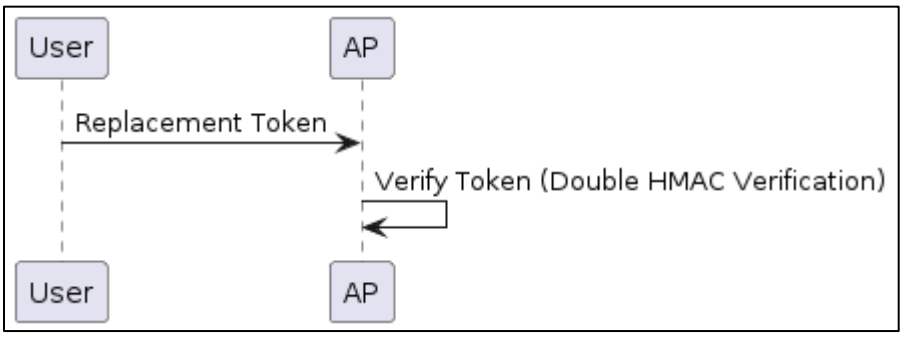

Figure 2. High level replacement protocol.

Boot

The boot tool communicates with the application processor (AP) and attempts to boot the system. The host computer issues the command "boot" to the AP. Upon receipt, the AP validates all components by starting a TLS-Lite session with each component – this entails initiating a TLS-Lite handshake with each component. If the TLS-Lite handshake is successful, then the component is valid (see SR1 for more details) and the component shall immediately boot. If either of the TLS-Lite handshakes fail, the boot process will be aborted, and the AP will not boot.

After the TLS-Lite handshakes are completed (and the components are booted), the AP will boot itself. The TLS-Lite session is not ended after the AP boots, however – the session key is retained for post-boot secure communication. The protocol is also shown in Figure 3 below.

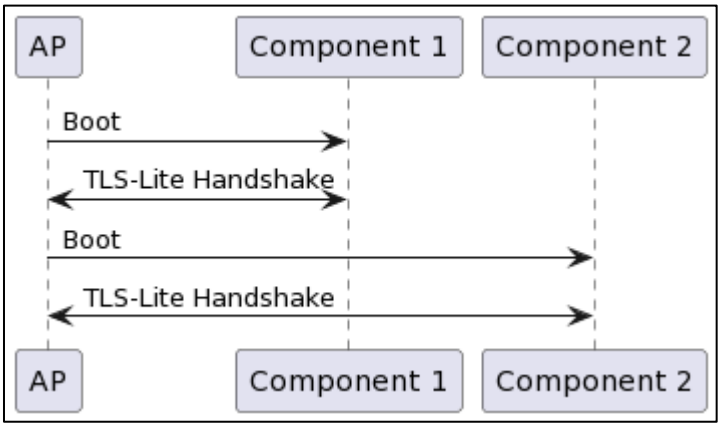

Figure 3. High level system boot protocol.

Secure Send and Receive

After the boot process is complete, a secure communication channel is established via the TLS-Lite handshake that was completed during the boot procedure. The secure communication channel uses the shared symmetric key generated during the TLS-Lite handshake in order to maintain message secrecy and authenticity. Messages are prepended with a running count of all messages sent previously and are encrypted and sent along with a MAC (HMAC) of the entire message (including the count).

Threat Modeling

Attackers may use a variety of methods to attack the MISC system. This design is built to defeat the attacks considered to be high risk. These potential threats are listed here in order of their risk (high to low).

Buffer Overflow Attacks

The MISC system takes input from both a host computer as well as different components/application processors. Thus, buffer overflow attacks may be possible on naïve

implementations of the system. This can open the system up to code injection attacks and more that can leak secret information such as attestation data and encryption keys.

These attacks can be protected against by carefully implementing bounds checking on any buffers and not allowing unbounded input to an unbounded buffer.

Replay Attacks over I2C

The reference design issues commands to the component over an insecure communication channel: I2C. A secure design must be resilient to attackers resending messages on the bus or otherwise interfering with this communication channel. This means that information must be checked for privacy and authenticity.

These attacks can be protected against by using TLS 1.3, which generates unique session keys in each handshake. Furthermore, the TLS 1.3 record protocol protects against the injection of malicious messages onto the I2C bus.

Timing Analysis

The system requires the user to enter both a PIN and a Replacement Token in order for certain functions to work. However, an attacker without this information must not be able to recover it from the system. The reference design employs the built in `strcmp` function, which can be vulnerable to attacks on the timing side-channel. Since the function compares two strings until a difference is found, an attacker can deduce the correct string by entering several strings, noting that longer runtime is associated with more correct characters. Thus, any comparison functions must be resilient to the timing side-channel.

These attacks can be protected against by using a side-channel resilient comparison function (such as Double HMAC Verification). Such a function will run in a constant time for any comparison of strings of length n. In other words, for all comparisons of strings of a certain length, the function will run for the same amount of time (approximately), thereby not revealing information about the secret information being compared with.

Power Analysis

The system has various functions that operate on private data, most notably, encryption and decryption functions. Power Analysis (more likely differential power analysis (DPA)) can be used to extract information about cryptographic keys and plaintexts. Thus, these functions must use implementations that are resilient to such side-channels (as well as others not mentioned here).

These attacks can be protected against by using side-channel resilient implementations of cryptographic functions. Many cryptography libraries targeting embedded systems are written with this in mind.

Fault Injection

The MISC system may be vulnerable to fault injection attacks such as power glitching, clock glitching, electromagnetic fault injection, etc.. These attacks are complex to perform and are thus considered low risk due to the nature of the competition. However, they still may have the potential to defeat the security protections detailed in this document (by skipping critical authentication processes, for example).

Security Requirements

Security Requirement 1

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

The boot process validates components via the TLS-Lite handshake. Essentially, the TLS-Lite handshake involves the exchange and verification of device certificates in order to securely agree upon a shared symmetric key. The specific configuration will use RSA, ECDH, AES-128 (due to on-board hardware accelerators), and SHA-256. Once the handshake has been completed successfully with each component, the AP shall boot.

Security Requirement 2

Components should only boot after being commanded to by a valid AP that has confirmed the integrity of the device.

Similar to SR1, SR2 is fulfilled via the TLS-Lite handshake that occurs as part of the boot process. The handshake will be configured to use the mutual authentication variant so that both the server device (AP) and client devices (components) authenticate each other's certificates. Thus, the TLS-Lite handshake will fail if either the AP or the component does not possess a valid certificate (and the associated private key).

Security Requirement 3

The Attestation PIN and Replacement Token should be kept confidential.

The Attestation PIN and the Replacement Token shall be kept secure via the use of a side-channel resilient comparison function. This is implemented in the design with the Double HMAC Verification protocol. When attempting to verify a user-inputted PIN, the system will compute an HMAC of the input and the real PIN using a randomly generated ephemeral key. The HMACs will be compared in order to determine validity without leaking information about the PIN over the timing side-channel.

Security Requirement 4

Component Attestation Data should be kept confidential.

Component Attestation Data is only released via a successful Attest command (on the AP). This means that the component is communicating with a valid AP, which has successfully confirmed the accuracy of the user's PIN. After authentication via a TLS-Lite handshake, the attestation data is sent (over an encrypted TLS-Lite channel) to the AP. This ensures the secrecy and authenticity of the attestation data.

Security Requirement 5

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

The boot procedure entails the AP starting a TLS-Lite session with each of the components. This serves to authenticate all devices that are paired with it. Message integrity, authenticity, and privacy are protected by the TLS Record protocol, which serves not only to obfuscate messages, but also to protect them from modification by a man-in-the-middle adversary. Use of both AES encryption and CMAC during communication fulfills SR5.

Protocols

TLS-Lite

TLS-1.3 is a complete, but secure protocol for authenticating devices and creating a secure communication channel in the open. However, TLS-1.3 is also very complex and embedded-device compatible implementations of the protocol can be difficult to port to specific devices/systems using specific I/O protocols. Ultimately, due to a lack of time and manpower, the full TLS-1.3 library implementation was scrapped and reduced down to a simplified custom protocol dubbed "TLS-Lite." The TLS-Lite protocol is described in detail below.

Disclaimer: While implementing your own cryptographic protocols is generally a bad idea due to the potential of introducing vulnerabilities related to the implementation, it was decided that completing a less secure protocol is more effective than failing to complete a more secure protocol.

TLS-Lite Protocol

The user of the MISC system interacts solely with the AP. Upon receipt of a relevant command (see above), the AP will initiate a TLS-Lite handshake. First, the AP sends a message to the target component commanding it to prepare for a TLS-lite handshake. Next, both devices will generate 128B of random data.

The component will send its information first: it sends its 128B of random data followed by its 32B public key (ECC), which is signed by the deployment (CA) at build-time. The AP receives this data, verifies the public key's authenticity (using its signature), and sends its signed public key and random data to the component (assuming the key was verified successfully). The Component receives this data and verifies the AP's certificate as well. Now, both parties participate in an Elliptic-Curve-Diffie-Hellman key exchange (ECDH), using each other's public (and verified) ECC keys. The exchanged Elliptic-Curve key pair is used to obtain a shared secret.

Both parties pass this shared secret into a KDF (the algorithm chosen here was HKDF) in order to derive a session key from the temporary Elliptic-Curve Diffie-Hellman key. The random data that was sent by the AP and Component is used as a salt for this key derivation. Finally, both devices compute a CMAC of the nonces exchanged between them in order to protect against adversaries injecting stale random data into the handshake. Both devices send each other their MACs and verify. If either of the device's MACs do not match, the protocol will fail. Otherwise, the session key is then used for any secure communication over the TLS-Lite channel.

If at any point the protocol fails, the device detecting the failure will issue a message to the other to abandon the handshake and all communication will cease. The high level protocol at hand is shown in Figure 4 below.

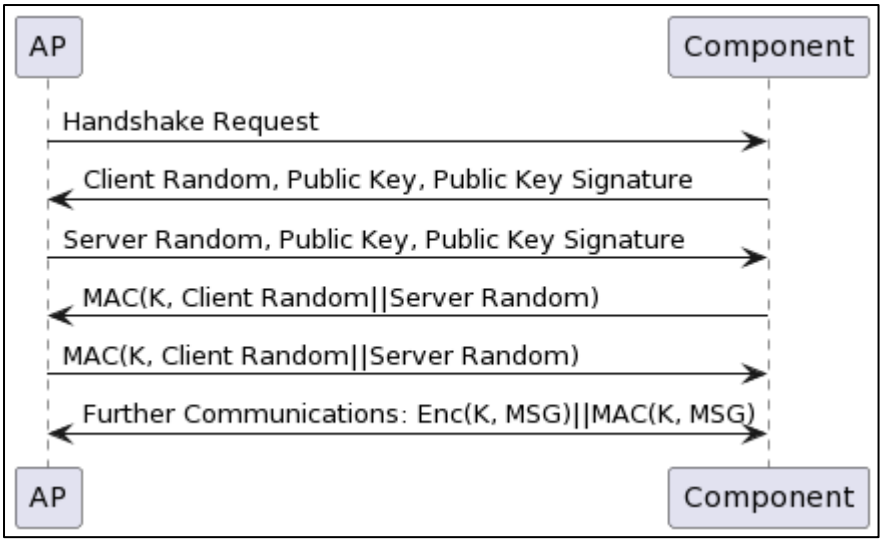

Figure 4. High level protocol of TLS-Lite, a simplified implementation of TLS.

Double HMAC Verification

In order to keep device secrets such as user PINS and replacement tokens confidential, it is necessary to use protocols that compare the data in constant time, as not to reveal information about their structure via the timing side-channel. For this reason, the MISC system applies the following protocol for comparing two secret strings.

First, the system generates a random key (in this case 32B) to compute the MACs with. Next, two HMACs are computed – one for each of the strings being compared. Finally, the two HMACs are compared with one another. Built in C functions such as `memcmp` can be used for the final comparison since the value of the HMAC does not reveal anything about the secret data being compared against. The high level protocol is shown diagrammatically in Figure 5 below.

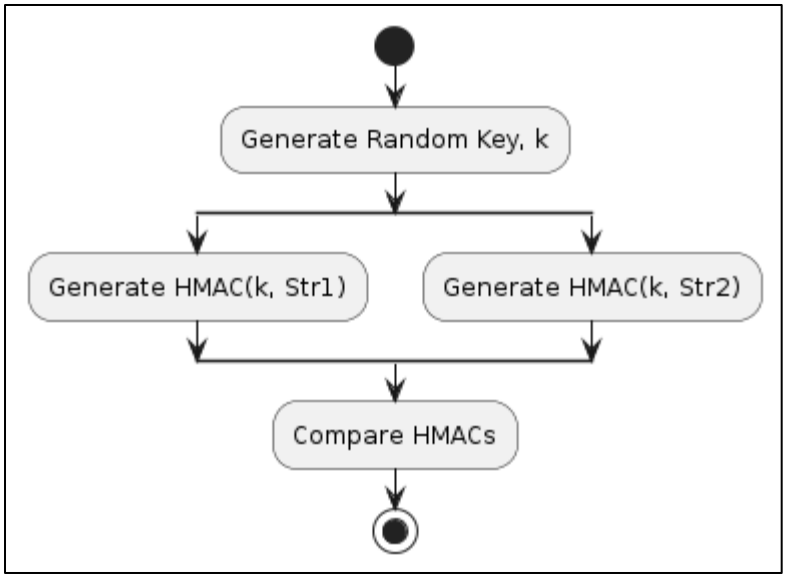

Figure 5. Double HMAC Verification Protocol.

Stretch Goals (Scrapped)

Some goals for the project were not easily reachable in a short timespan. Thus, these were left as secondary stretch goals should the time and bandwidth be available to implement them. Unfortunately, due to a lack of time and manpower, these goals were not completed.

RISC-V Coprocessor

Attackers may use side-channel attacks to gather information about the system and its data, including confidential secrets and encryption keys. Since the MAX78400-FTHR is a dual-core system, featuring both an ARM and RISC-V processor, the RISC-V processor (since it operates at a lower clock speed) can be used to generate side-channel noise to inhibit side-channel analysis. Simply executing random computations from startup should be sufficient to generate random-looking noise. While SCA can ultimately still defeat this protection, many more traces will be needed in order to extract useful information.

The RISC-V coprocessor can also serve as a watchdog unit in order to protect against fault injection. The processor shall take hashes (MD5) of the FLASH memory section. This serves to protect against code injection attacks. In addition, the coprocessor can monitor the progress of the main (ARM) processor in critical sections. The exact protocol for this has not yet been decided.

Non-Volatile Memory Probing Protection

Attackers may use a variety of methods to probe non-volatile memory in order to collect secrets and encryption keys. A stretch goal is to find a method of protecting (at least parts) of non-volatile memory in order to keep certain information private.

Appendix

Appendix A: Revisions

Revised the document to reflect TLS-Lite protocol, ECC usage, and added protocol sequence diagrams (4/9/2024).