CyberSecurity Club at San Francisco State University

Initial Design Document

Ethan Hanlon[a], Michael Petrossian[a]

[a]San Francisco State University, 1600 Holloway Avenue, San Francisco, CA 94132, USA

Abstract

This research focuses on designing and implementing a secure MISC device that will be built to adhere to the standards of the functional and security requirements. Our design will aim to withstand all attack scenarios presented by the MITRE documentation, as well as provide a new approach to embedded system programming with the popular and recent memory-safe Rust programming language.

1. Introduction

The aim for this project is to create a device that can communicate securely and validate components within itself for integrity and authenticity. In developing and implementing a MISC secure medical device, our team found that cryptography provided the strongest protection against most attack scenarios and fulfilled much of the requirements that demanded confidentiality, integrity, and authenticity. Although our design appears bullet-proof, there are still some fundemental flaws that, as will be discussed later, can only be mitigated rather than completely resolved. Our methods of encryption will comprise of elliptic curve key pairs, psuedo-random token seeds, and hash digests - all of which can be generated and derived from the build deployment.

2. Functional Requirements

2.1. Build Environment

For the purpose of preventing bugs and keeping safe coding standards, the Rust programming language will be used. This compromise will involve improvising the current build environment used in the original design reference that was given to the team by the organizers.

2.2. Attest

2.3. Replace

2.4. Boot

2.5. Secure Communication

3. Security Requirements

3.1. Security Requirement 1

Security Requirement 1 (**SR1**) requires us to ensure the Application Processor will not boot unless expected components are present and valid. Our plan for this involves a two-pronged approach which uses public/private key encryption and a secure cryptographic nonce.

During the build process, we will generate a public/private key pair. The public key will be flashed onto the components, and the private key will be stored in the application processor. Additionally, a random seed will be generated using /dev/urandom and stored in the application processor. This seed will be used by the components and the application processor to generate and validate cryptographic nonces.

The components will use the public key to encrypt the nonce along with their own unique identifier and send it back to the application processor over I2C bus. The application processor will use the private key to decrypt the nonce and verify that it matches the original nonce. If the nonces match, the application processor will know that the components are valid and will continue with boot. If no message is received, or the nonces do not match, the application processor will not boot.

3.2. Security Requirement 2

Security Requirement 2 (**SR2**) requires us to ensure that the components will not boot until commanded to do so by an Application Processor that has validated the integrity of the system. Our plan to meet this requirement is much the same as the plan for Security Requirement 1, but in reverse.

During the build process, a random seed will be burned onto the firmware as delineated in SR1. In addition, each component will receive a public/private key pair similar to the AP's. This key will be used to encrypt messages before transmitting them over the I2C bus, as well as to verify digital signatures.

After the application processor verifies the message sent by the components as delineated in SR1, it will return an acknowledgement, encrypted with the component's public key. Upon receipt of this acknowledgement, the component will use its private key to decrypt the message and validate its digital signature. From there, the nonce will be checked against the random key generator. If the message is not present, the digital signature is invalid, or the nonce verification fails, the component will immediately halt the boot process. Conversely, if all checks pass - suggesting the AP successfully validated the components - the component will proceed with the boot sequence.

3.3. Security Requirement 3

Security Requirement 3 (**SR3**) Requires that the Attestation PINs and Replacement Tokens are kept confidential. To accomplish this, we will hash and salt the secrets to prevent the plaintext versions from being exfiltrated. It's unrealistic, given physical access to the machine, to expect that we can fully prevent the memory holding the secrets from being extracted from the device. However, by using SHA256, we can make the data useless to the attacker. By salting the PINs using a special token generated at compile time, we can prevent the use of a rainbow table to get around the hashing. Only an authorized personal with the secret PIN is able to extract the attestation data.

3.4. Security Requirement 4

Security requirement 4 (**SR4**) states that the attestation data is to be kept confidential and secure from unauthorized changes. Assuming that an attacker has the ability to modify the data directly, fulfilling this requirement can be done by encrypting the attestation data with the public key of the AP and embedding the data with its hash digest. Whenever an authorized user requests the attestation data, the AP first decrypts the data received by the component and then verifies the integrity by matching the resulting hash of the data with the embedded digest that went along

with it. This technique is inspired by modern message authentication, with message authentication codes (MAC) that ensure the integrity of data whenever in transit.

References

[1] S. Scholes, Discuss. Faraday Soc. No. 50 (1970) 222.

[2] O.V. Mazurin and E.A. Porai-Koshits (eds.),

Acknowledgements

This template was originally created by the Partnership for Advanced Computing in Europe (PRACE). It was modified for use by the San Francisco State University eCTF team.

Make sure to include acknowledgements for people who helped in the project, including professors, graduate students, and other team members.