Design Document

1. Outline of Host Tools, Bootloader, and Secrets

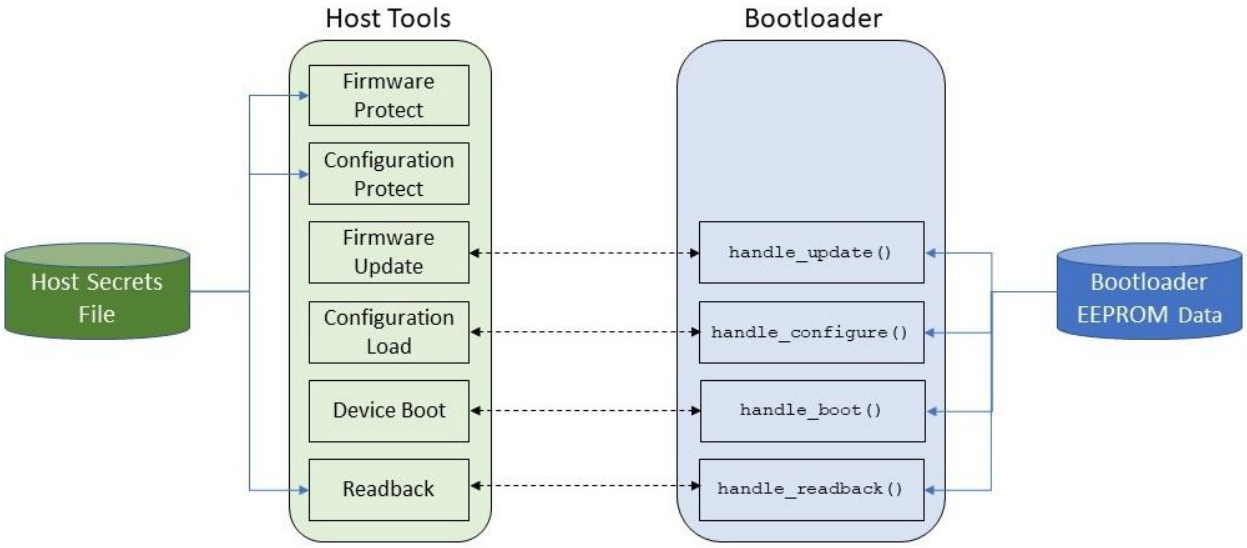

Figure 1: Design Structure

Figure 1 shows the structure of the code and the interactions between the host tools and the bootloader. The host tools are written in Python and they execute on a linux host. The bootloader is written in C and executes directly on the ARM-based target board, without an operating system. The host tools include Python files to implement each of the required functions as specified in the rules document. The bootloader is implemented in a single main file, bootloader.c, and multiple other files containing support functions. The bootloader continually checks for commands from the host tools and calls the appropriate handler function to execute each command. The dashed arrows in Figure 1 show the communication between functions in the host tool and their corresponding handler functions in the bootloader.

Both the host tools and the bootloader have secret keys which are used to support some of their functions.The host tool secrets are stored in the Host Secrets File shown in Figure 1, and the bootloader secrets are stored in the Bootloader EEPROM Data, also shown in Figure 1. The blue arrows indicate which host tool functions have access to the Host Secrets File, and which handler functions in the bootloader have access to the Bootloader EEPROM Data.

The host tools and the bootloader communicate via a UART connection which is accessed directly by the bootloader and accessed using a TCP socket by the host.

2. Satisfying Security Requirements

Several design decisions were made to satisfy the security requirements stated in the rules document. We enumerate each requirement and describe the security techniques used in the design as a whole to satisfy each requirement. We also list each host function and describe how the host tools and bootloader are modified to implement the security techniques used.

2.1 Security Requirements

Confidentiality - Symmetric encryption is used to protect the firmware and configuration images. We will disable the debug interface of the board so that the attacker cannot extract the bootloader, firmware, or configuration at runtime, or control the execution of either the bootloader or the firmware.

Firmware/Configuration Integrity and Authenticity - Digital signatures are associated with the firmware and configuration images so that the bootloader can verify their integrity and authenticity before firmware update, configuration load, and boot..

Firmware Versioning - The firmware version number is compared to the new firmware version number before allowing the new firmware to be installed. Both the version numbers are signed with their associated images to guarantee their integrity.

Readback Authentication - The Challenge Handshake Authentication Protocol (CHAP) is used so that the bootloader can verify the authenticity of the technician.

2.2 SAFFIRe Operations

Each host tool function and corresponding bootloader handling function has been enhanced to satisfy the security requirements specified in the rules document. Here we enumerate each host tool function and we specify how it has been modified to satisfy the security requirements.

Firmware Protect - This function uses symmetric encryption to produce a protected firmware image and ensure the confidentiality requirement. The rules state that it should not be possible for the attacker to modify the version number of a protected image, so the firmware version number is encrypted with the firmware image. A digital signature is created and associated with the protected firmware image and version number. The digital signature will be needed to verify the integrity and authenticity of the image by the bootloader.

Configuration Protect - This function uses symmetric encryption to produce a protected configuration image and ensure the confidentiality requirement. A digital signature is created and associated with the protected configuration image. The digital signature will be needed to verify the integrity and authenticity of the image by the bootloader.

Firmware Update - The protected firmware image is sent to the bootloader. The handle_update() function in the bootloader verifies the digital signature before writing the protected firmware image into the flash memory. If the signature verification is successful then

the protected (encrypted) version of the firmware is written into the flash memory. The firmware in flash is encrypted to ensure its confidentiality. The firmware version number is extracted from the protected image and is compared to the version number of the current firmware, or to the highest previous non-zero version number if the current version number is 0. The firmware only gets written if the new version number is equal to or greater than the compared one. This policy applies up to version 0xFFFF.

Configuration Load - The protected configuration image is sent to the bootloader. The handle_configure() function in the bootloader verifies the digital signature before writing the protected configuration image into the flash memory. Only the valid configuration images made by the Configuration Protect host tool will be accepted. If the signature verification is successful then the protected (encrypted) version of the configuration is written into flash memory. The firmware in flash is encrypted to ensure its confidentiality.

Device Boot - The handle_boot() function in the bootloader verifies the digital signatures of the firmware and configuration images in flash, in order to satisfy the integrity and authenticity requirements. If verification is successful then the firmware and configuration images are decrypted and written into the appropriate locations in SRAM and flash. The bootloader then jumps to the start of the firmware to begin execution.

Readback - In order to satisfy the readback authentication security requirement, the Challenge Handshake Authentication Protocol (CHAP) is used before either the firmware or configuration data is transmitted. The CHAP protocol proceeds as follows:

1. We ensure that the host computer and avionics device share a secret key, K.
2. The bootloader handle_readback() function sends a pseudo-random string to the host, in the clear.
3. The host concatenates the string with the shared secret key K and computes the hash of the concatenation.
4. The host sends the hash back to the bootloader.
5. The bootloader also concatenates the string with the shared secret key K and computes the hash of the concatenation.
6. The bootloader compares the hash which it computed to the hash received from the host.
7. If the hashes are the same then authentication is successful.

If authentication is successful then the firmware or configuration data is decrypted and transmitted to the host.

3. Defending Against Hardware Trojans

Hardware trojans can give the attacker access to flash memory which contains the firmware, the configuration data, and the bootloader. The firmware and configuration data are encrypted in flash, so they cannot be directly accessed by hardware trojans. This is because reading

ciphertext yields no useful information and either writing or erasing will be detected when the digital signatures are verified at boot time. The only contents of the flash which need additional protection from hardware trojans is the bootloader, since it is not encrypted. The bootloader does not contain secrets, so reading the bootloader is not a problem. The attacker also cannot erase the bootloader without setting an entire 1K block, which would cause the bootloader to fail in an obvious way. However, the attacker has great freedom to change the behavior of the bootloader by writing to change the instructions and operands of its instructions.

The bootloader will be protected in multiple ways. Firstly, during the system build process we will ensure that the relative locations of the bootloader functions are shuffled so that the attacker cannot know where each instruction is before the trojan runs.

The bootloader cannot verify it has not been modified by the trojan during the bootstrapping process, so the host tools will be able to complete an interactive procedure to verify that the trojan has not modified the bootloader. This procedure will consist of a random challenge issued by the host tools containing a string. The bootloader will compute a cryptographic hash on its own code using the provided challenge as a salt and reply to the host tools with the hash. The host tools can compute the same hash and if there is a mismatch, refuse to interact with the bootloader.

During runtime, all interactions with flash will be in code running in SRAM. This will prevent the trojan from modifying currently running code. Memory protection using the Cortex-M4 processor's Memory Protection Unit will also be enabled so code cannot be executed from storage regions. In addition, before and after operations that could trigger the trojan are performed, the code will compute a checksum of the bootloader region and verify that it has not changed.