Challenge Description and Rules:

SAFFIRe: a Secure Avionics Flight Firmware Installation Routine

Contents

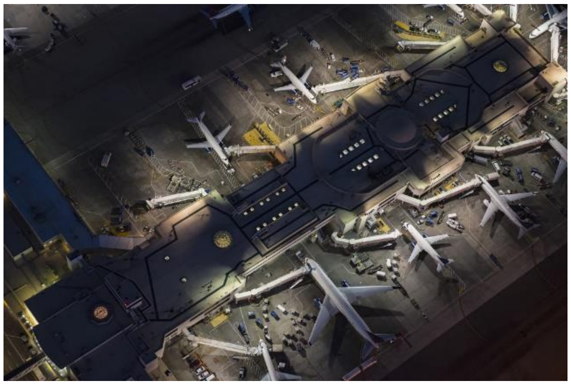
1 Challenge Overview

1.1 Motivational Scenario

You are part of an elite design and development team at a startup company developing firmware for electronic devices that are used in aircraft (i.e., avionics). The company has had success developing cutting-edge navigation algorithms that will be used worldwide to improve aircraft flight timeliness and cost. However, you have hit a major roadblock during prototyping and testing. You've discovered that once your devices ship to airports all over the world you will need to periodically update the in-flight firmware and allow customers to load in flight-specific configurations. How will you support this functionality and ensure that the system is secure? You and your team have been tasked with figuring it out! We need a secure update system – the CEO already picked out the name: the **"Secure Avionics Flight Firmware Installation Routine" (SAFFIRe)**.

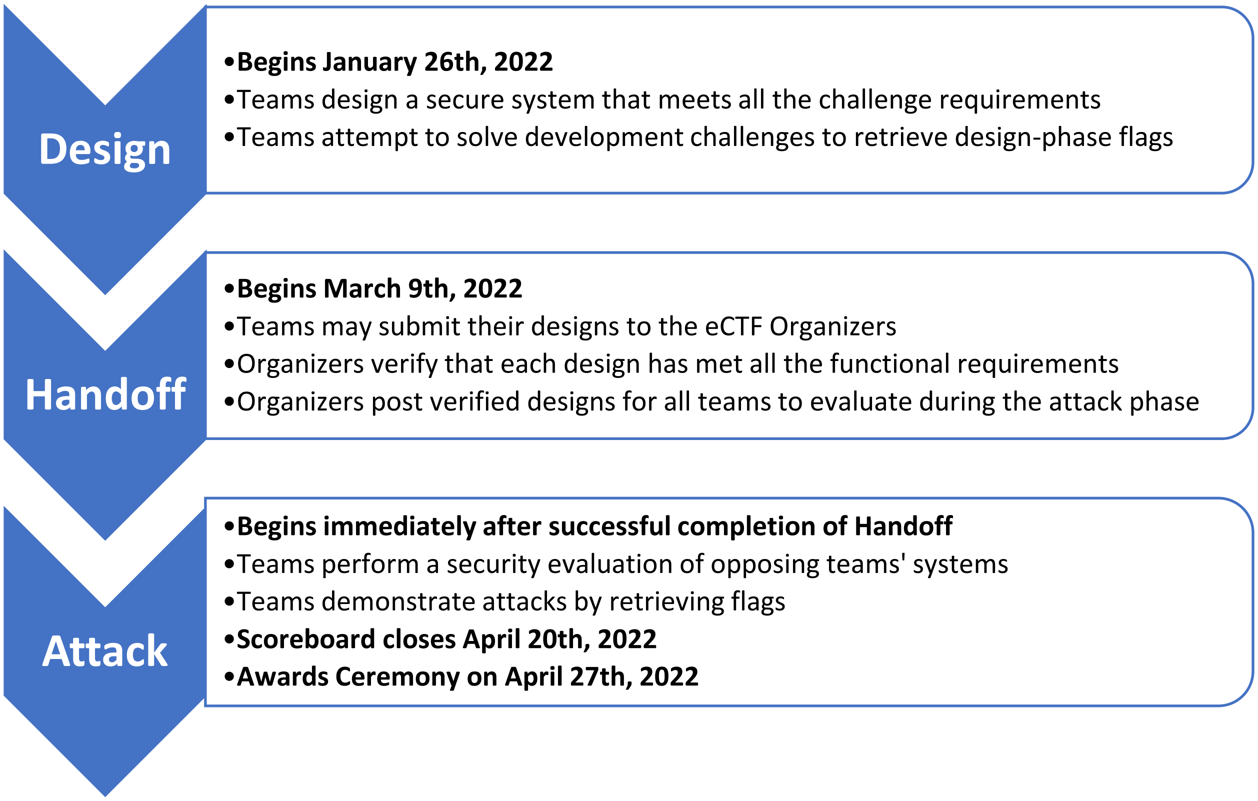
SAFFIRe consists of two parts: 1) the SAFFIRe bootloader for the avionic device and 2) the SAFFIRe host tools. The SAFFIRe bootloader does not run during flight and is instead responsible for installing and launching trustworthy firmware and configurations. Additionally, the bootloader must support a readback mode that lets a legitimate technician request information from the current firmware and configuration. The SAFFIRe host tools allow technicians to package updates and then communicate with the bootloader to install updates and request debug information from the device. Be careful though, as disgruntled employees working in aircraft ground crews may try to install their own malicious firmware and extract your proprietary algorithms, flight configurations, and sensitive device data.

Unfortunately, there's one more wrinkle to worry about… a recent news article revealed that your hardware manufacturing partner might not be as trustworthy as you'd like. As a result, some of the hardware that you rely on may contain malicious modifications that impact the security of your system if you aren't careful.



1.2 Competition Phases

This is a design-build-attack competition with phases for both attack and defense:



Design

- **Begins January 26th, 2022**
- Teams design a secure system that meets all the challenge requirements
- Teams attempt to solve development challenges to retrieve design-phase flags

Handoff

- **Begins March 9th, 2022**
- Teams may submit their designs to the eCTF Organizers
- Organizers verify that each design has met all the functional requirements
- Organizers post verified designs for all teams to evaluate during the attack phase

Attack

- **Begins immediately after successful completion of Handoff**
- Teams perform a security evaluation of opposing teams' systems
- Teams demonstrate attacks by retrieving flags
- **Scoreboard closes April 20th, 2022**
- **Awards Ceremony on April 27th, 2022**

1.2.1 Design Phase

In the Design Phase, each team must design and implement a system that meets a set of functional requirements and security requirements. **Teams will be provided with a reference design that meets the functional requirements but intentionally does not meet any security requirement**. Teams may use the reference design as a starting point or build their design from scratch. In either case, the directory structure of the submitted design must match the structure defined in the Technical Specifications Document.

During the Design Phase, teams may score points by capturing Design Phase Flags which show that teams are making progress towards a complete design. Flags must be submitted by their deadlines for points to be awarded.

New This Year: In addition to the reverse engineering challenges and bug bounty program introduced last year, teams may score points by completing one or more of a series of emulated side-channel analysis challenges. See Section 6.1 for details

1.2.2 Design Handoff

Starting March 9th, each team may submit their completed design to the organizers. The organizers will then verify that their submission meets all functional requirements. If a submitted design passes functional testing, that team moves into the Attack Phase. Therefore, the date and time of transition from Design Phase to Attack Phase may vary between teams. For example: If Team A and Team B both submit systems on the Handoff date, but only Team A's system passes the tests, then only Team A will move into the Attack Phase while Team B remains in the Design Phase until they submit a system that meets all functional requirements.

Each submission should include all source code and documentation for the submitted design. This includes all code necessary for building and running the system in accordance with the system functional requirements. The system source code (and optionally the documentation) must reside in a Git repository, and any extra documentation not stored in the repository must be posted along with the submission request on the team's official MITRE slack channel.

Upon receiving a submission, the eCTF organizers will clone and provision the team's system via their Git repository. Then, the organizers will run a sequence of test cases that validate whether the system meets the functional requirements. ***Note**: **The test cases will not check any security requirements**.* The eCTF organizers will contact the submitting team within two (2) **business days** after the submission indicating whether the system is accepted or not.

1.2.2.1 Accepted Designs

If a system is accepted, the organizers will inform the team and create a Handoff package that includes all source code, all documentation, and all distributed Attack Phase artifacts (See the Technical Specifications Document for more details). The team must approve of the Handoff package before advancing into the Attack Phase. ***Note:** Teams are not allowed to modify their designs after reviewing the Handoff package. The Handoff package serves as the final opportunity for teams to verify that they have not left any sensitive system materials in their repositories that they do not wish to be publicly known. Any changes to the design or functionality of the submission will require going through the full Handoff process (i.e., functional testing) again before moving to the Attack Phase.*

1.2.2.2 Rejected Designs

If a system is not accepted, the eCTF organizers will inform the team and provide an explanation for why the design did not pass testing. The submitting team must then revise their design and submit a new version to the organizers.

1.2.2.3 Automated Design Testing

New This Year! The eCTF organizers will provide teams with access to an automated testing server that they can use to run their systems through test cases before doing a formal submission. This is intended to improve the turnaround time for getting test results and provide teams with a way to better-validate their designs before submitting to the eCTF organizers. More details on how to use the testing server will be provided during the Design Phase.

1.2.3 Attack Phase

During the Attack Phase, each design that has been validated during Handoff will be made available to other teams in the Attack Phase for attack. Teams will be able to attack other teams' designs in two ways: On a physical microcontroller and/or on a server with an emulated microcontroller and aircraft simulation.

1.2.3.1 Materials and Information Available to Attackers

For each design, the files below will be made available to attacking teams

- All source code (with the .git directory removed)
- The most recent documentation provided to the eCTF organizers
- Firmware and Configuration files protected via the design's protect tool
- A protected copy of the design's SAFFIRe bootloader to load onto a keyed microcontroller
- Access to a server running an emulated device loaded with the design's SAFFIRe bootloader
    - An interface to communicate with the emulated device's serial port
    - An interface to collect side-channel traces from the emulated device
    - An interface to trigger resets in the emulated device
    - An interface to inject hardware trojans

Note: *The SAFFIRe bootloader running on the physical and emulated devices will be identical so that teams can use attack results obtained from one platform towards extending the attack on the other. Certain flags will only be accessible by attacking the emulated device (See Section 6.2 for more details).*

2 System Overview

2.1 System Components

The system is broken into three main components: the host computer, the avionic device, and the aircraft. The SAFFIRe host tools run on a Host Computer and talk to the SAFFIRe bootloader installed on the avionic device. Once the ground crew installs a firmware and flight configuration onto the avionic device using the host tools and the bootloader boots the firmware, the avionic device is placed on an aircraft and the firmware communicates with other devices on the avionics bus.

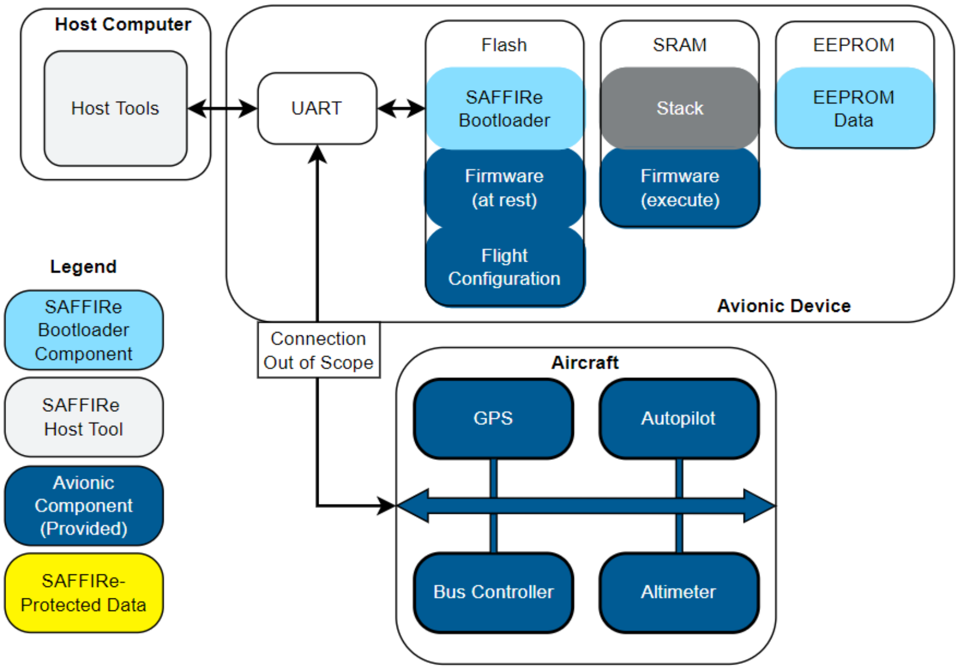


Figure 1 Avionic System Architecture

2.1.1 Challenge Platform

2.1.1.1 Host Computer

The host computer is a general-purpose computer used to protect firmware and configuration images, perform updates with the SAFFIRe bootloader, and read data back from the bootloader. Since the host computer is a general-purpose computer, the host tools running on it will have access to common Linux programs, utilities, and packages. The reference design provided by the eCTF organizers implements the host tools in Python, but teams are allowed to use a programming language of their choice. ***Note: if a team wishes to use a language other than Python, please consult the eCTF organizers for approval before implementing the host tools.***

2.1.1.2 Avionic Device

The avionic device uses a Texas Instruments Tiva-C Microcontroller (TM4C123GH6PM)[1]. Teams will develop their SAFFIRe bootloaders to run on this microcontroller. Since both emulated and physical copies of the microcontroller will be used throughout the competition, some of the hardware peripherals are not supported. Please see the Technical Specifications Document for a detailed list of which peripherals are supported.

2.1.1.3 Avionics Bus

Teams are not responsible for developing any part of the avionics bus and all firmware and configuration binaries that communicate over the bus will be provided by the eCTF organizers. The firmware is only able to communicate with the avionics bus in the emulated environment.

2.1.2 Development Resources

Teams will be provided with the following resources:

Teams will receive an un-keyed Tiva-C TM4C123G Launchpad[2] to use as a development microcontroller. Additionally, instructions will be provided to run the host tools on a local computer to test the entire SAFFIRe system using physical hardware. The development microcontroller can also be used during the Attack Phase to practice and prototype attacks before performing them on the keyed microcontroller provisioned with a secure design. This device will not be able to run Attack Phase SAFFIRe bootloaders provisioned by the organizers.

Teams will receive a keyed Tiva-C launchpad which teams can use to load MITRE-protected opponents' SAFFIRe bootloaders during the Attack Phase. This device will be unusable during the Design Phase.

Finally, teams will be given access to a development server loaded with all necessary programs for running the system using emulated hardware. The emulated microcontroller will run in a Docker container and communicate with the host tools through a local socket.

2.2 Avionic Device Lifecycle

This challenge encompasses the full lifecycle of an avionic device, ranging from microcontroller fabrication to its deployment on an aircraft for multiple flights. Figure 2 depicts the full lifecycle, with each stage explained in the following sub-sections. ***Note:** This sub-section is included to provide context*

[1] Tiva-C Microcontroller User Guide: https://www.ti.com/lit/pdf/spms376
[2] Tiva-C Launchpad User Guide: https://www.ti.com/lit/pdf/spmu296

about the avionic device's role in an aircraft flight and explain where and when attackers will have access to the device.

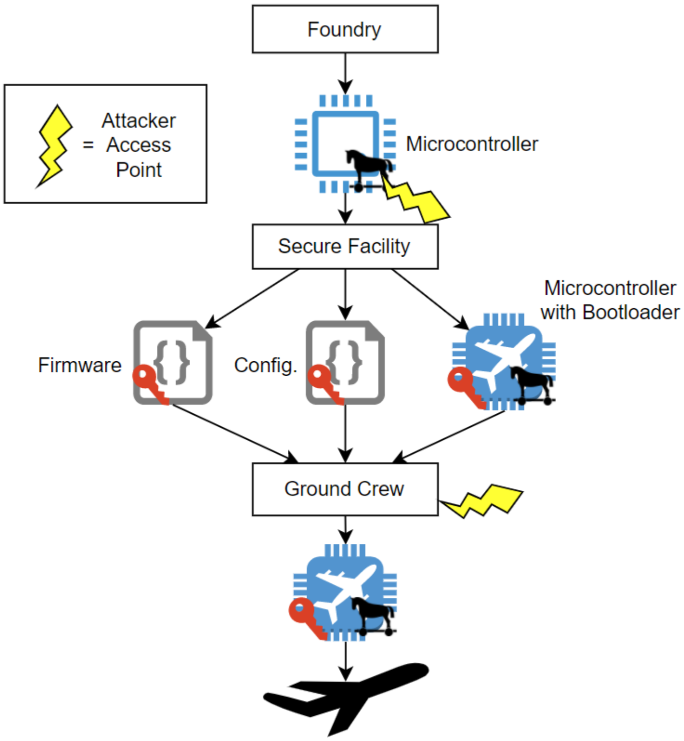


Figure 2 Avionic Device Lifecycle

2.2.1 Device Fabrication – Trojan Insertion

The device microcontroller is mass-fabricated at an untrusted foundry. Adversaries working at the foundry can insert trojans in the microcontroller hardware which may try to exfiltrate sensitive information or bypass software security checks to cause catastrophic behavior. This means that the SAFFIRe bootloader must be designed to ensure that data and code stored in the microcontroller are not tampered with. More details of the hardware trojan can be found in the Technical Specifications Document.

2.2.2 SAFFIRe Device Creation

Your company purchases an avionic device microcontroller from a supplier and then loads it with your SAFFIRe bootloader. This step is considered secure since the process of building of the SAFFIRe bootloader and loading it into the microcontroller are performed by trusted employees in a secure location.

2.2.3 Firmware and Configuration Image Protection

Your company also provides flight firmware and configuration files to be installed on the avionic device. The firmware is code that runs on the device during flight and performs critical algorithms and communicates with other devices on the avionics bus. The flight configuration contains flight-critical data such as coordinates and flight plans. A separate engineering team (i.e., the eCTF organizers) creates the raw firmware and configuration files, which are then protected by your SAFFIRe firmware/configuration protect tool. Adversaries are ***not*** able to interfere with the firmware and configuration protection process.

2.2.4 Aircraft Depot Updates

The protected firmware and configuration images provided by your company are sent to an aircraft depot along with the device containing the securely installed SAFFIRe bootloader. There, the ground crew uses SAFFIRe host tools to provision the device with the protected firmware and the configuration for a given flight. Unfortunately, there are untrusted staff working at the depot who may try to extract confidential data from the protected firmware and configurations and maliciously modify the avionic device before an aircraft flight. These adversaries have direct access to the device and may attempt physical and proximal attacks including power side-channel collection, fault injection, and hardware tampering.

2.2.5 Aircraft Flights

Once the ground crew place the provisioned avionic device on the aircraft and a flight begins, the SAFFIRe bootloader hands off control to the avionic firmware, which interacts with other devices on the avionics bus for the duration of the flight. As the device is installed onto the plane, the adversaries ***do not*** have access to the device during flight.

3 Functional Requirements

This section defines the functional requirements of your SAFFIRe design. *Figure 3* shows a typical flow for using the system, starting with building the system, and ending with repeated device boots, firmware and configuration updates, and device readback requests. All functionality listed must be implemented for a design to be accepted into the Attack Phase. These descriptions, along with the security requirements, are meant to help teams conceptually design their SAFFIRe system at a high level without worrying about implementation and code specifics. *Details on specific command-line arguments and implementation-based requirements can be found in the Technical Specifications Document.*

The phrases “must” or “must not” denote that the specified functionality is mandatory for a design to pass validation, the phrases “should” or “should not” denote that the specified functionality is encouraged and has security implications, and the phrases “may” or “does not need” denote that the specified functionality in that case is undefined and may or may not have security implications.

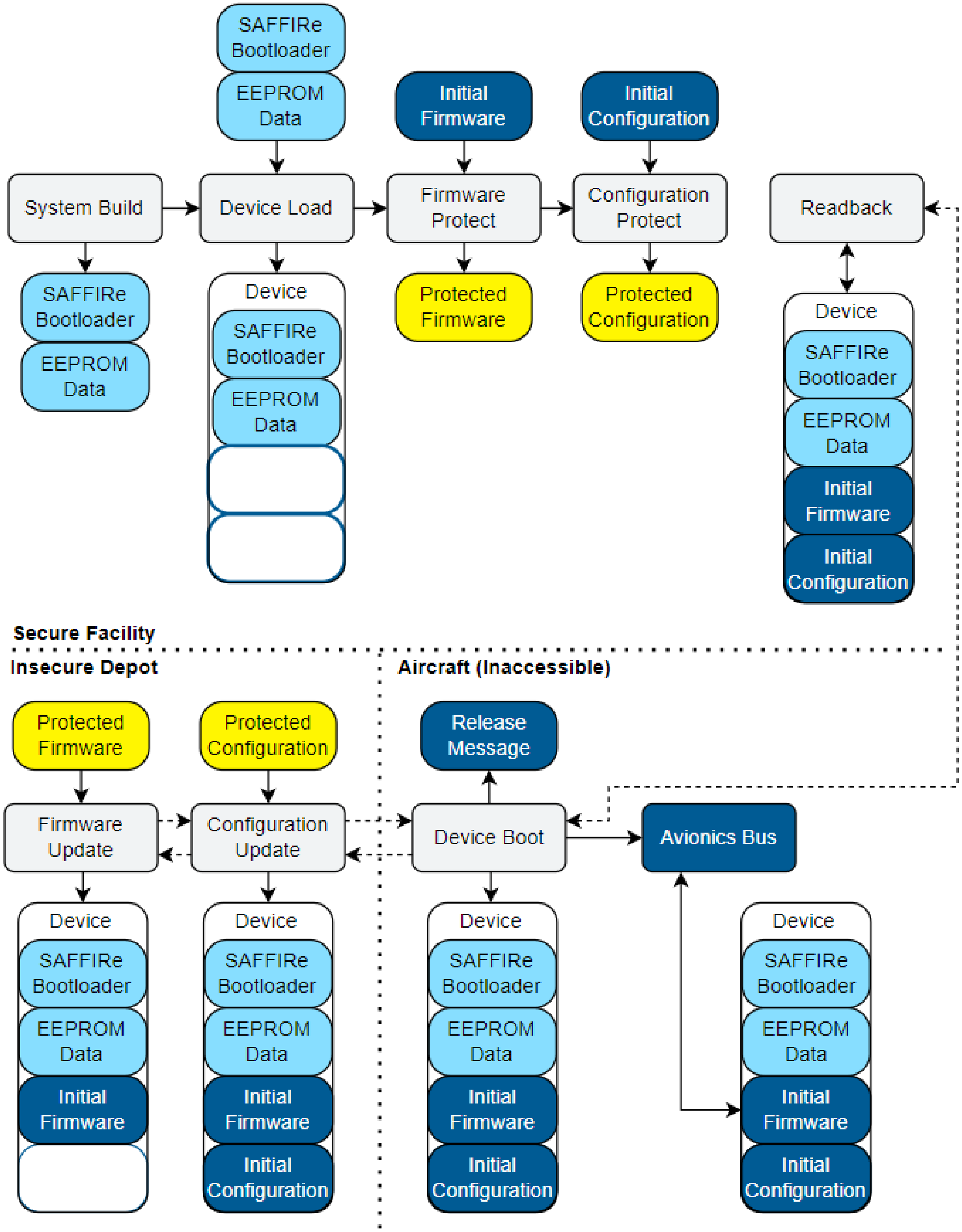


Figure 3 Typcial SAFFIRe Usage Flow

3.1 System Build

A full SAFFIRe system is built from the design source code. The build process is done in a secure environment that is inaccessible to the attackers and produces 3 main components that will not be released to attackers[3]:

- **Bootloader Binary** - The bootloader binary contains the full bootloader program that runs on the microcontroller. This file may have secrets built into it to enable the bootloader to verify and produce cryptographic material and other secret data.
- **Bootloader EEPROM File** – The EEPROM file can contain additional data or secrets that the bootloader may need to operate correctly.
- **Host Secrets File** – The host secrets file can contain secret data required for running the firmware and configuration protection tools and the readback tool.

The build process accepts an *oldest allowed firmware version* argument that specifies the initial firmware version that should be installed on the device (see Section 3.3.3.1).

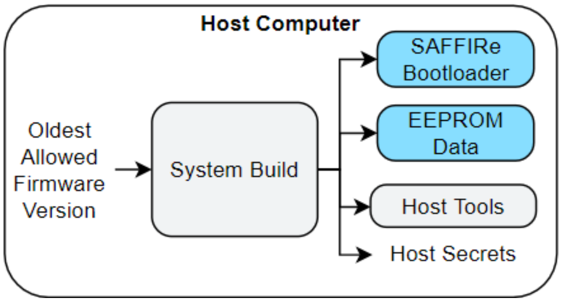


Figure 4 System Build Step

3.2 Device Load

After building the SAFFIRe system, the SAFFIRe bootloader and EEPROM contents are loaded onto the avionic device. Teams are not responsible for implementing the system load step and will be provided with tools to load their designs.

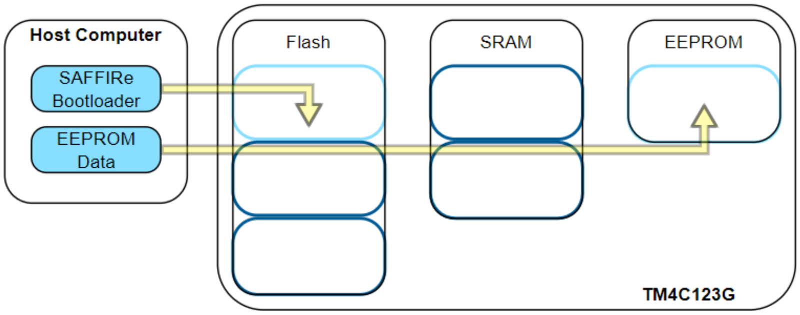


Figure 5 Device Load Step

[3] During the Attack Phase, attackers will be provided with MITRE-protected bootloader binaries and EEPROM files that will only be able to be loaded onto the keyed boards provided by the organizers. This secure load process is out of scope for attackers. However, the process of loading the SAFFIRe bootloader onto the emulated device is in scope for hardware trojan attacks.

3.3 SAFFIRe Operations

The following is a list of the primary functions of the SAFFIRe bootloader and host tools. The requirements for the each are explained in the following subsections.

- Firmware Protect – A firmware image, along with a version number and release message, is protected by the host tools and saved to a secure firmware image file.
- Configuration Protect – A flight configuration image is protected by the host tools and saved to a secure configuration image file.
- Firmware Update – A secure firmware image is given to the bootloader, which verifies its integrity and authenticity and installs it to the avionic device.
- Configuration Load – A secure configuration image is given to the bootloader, which verifies its integrity and authenticity and installs it to the avionic device.
- Device Boot – The bootloader verifies the integrity and authenticity of the installed firmware and configuration images, places them in plaintext at specific memory locations, and hands off execution to the firmware
- Readback – The bootloader sends the plaintext firmware and configuration images to an authenticated host tool

3.3.1 Firmware Protect

The host firmware protection tool must take a raw firmware binary (maximum of 16KB), version number (unsigned 16-bit value), and release message (max size of 1KB), and produce a protected firmware image that can be used with the firmware update tool. The firmware protection tool may access the host secrets file.

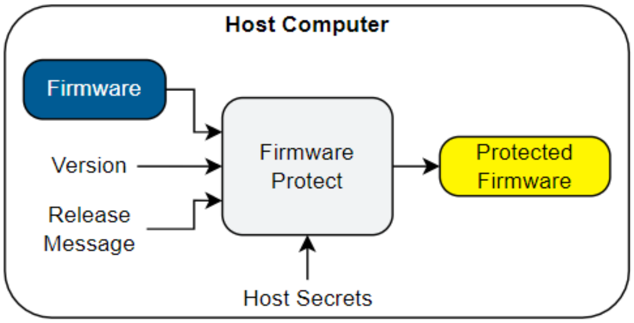


Figure 6 Firmware Protect Step

3.3.2 Configuration Protect

The host configuration protection tool must take a raw configuration binary (maximum of 64KB) and produce a protected configuration file that can be used with the configuration update tool. The configuration protection tool may access the host secrets. Unlike firmware, there is no version number or release message associated with flight configurations.

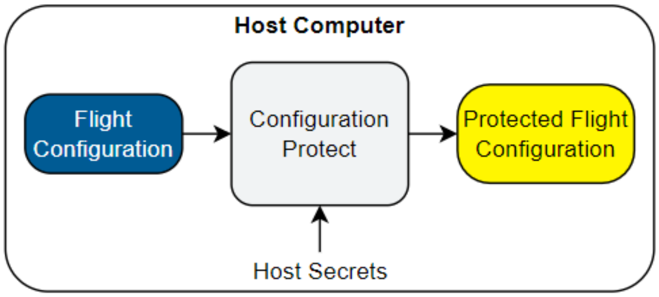


Figure 7 Configuration Protect Step

3.3.3 Firmware Update

The host firmware update tool must take a protected firmware image and send it to the bootloader for installation on the device. The tool is only required to perform updates with validly formatted protected firmware images and may return a non-zero exit code if any security violation is detected. This tool does **not** have access to the host secrets file.

The bootloader must accept a new protected firmware image from the host tools and install it to the device such that it can be booted later, including across power cycles. The firmware update routine should only install valid images created by the Firmware Protect host tool. If an update fails midway through, the bootloader must continue to operate correctly and meet the functional and security requirements. There are no requirements on how the firmware is stored on the device at rest. The bootloader must return to a loop that looks for host commands after performing a firmware update.

3.3.3.1 Versioning

Each firmware is packaged with a version number, and only firmware images with increasing version numbers are allowed to be installed per the following rules:

1. The initial bootloader build process must accept an *oldest allowed firmware version* option that indicates the minimum firmware version that should be accepted by the first firmware update the first time the SAFFIRe bootloader turns on. Your company is likely to build a SAFFIRe system for a new device, after multiple firmware versions have been released. New bootloaders should only install firmware updates starting with the version specified by the *oldest allowed firmware* version parameter.
2. The bootloader should only install firmware images with version numbers that are equal to or greater than the currently installed firmware version.
    a. Example: If firmware version 3 is currently installed, firmware images with version numbers 1 and 2 should not be installed, and firmware images with version numbers 3,4,5,6, etc. must be installable.
    b. One exception to this rule is a firmware image with version 0 which must be installable at any time.
        i. If firmware version 0 is installed, the bootloader should not install firmware images with version numbers lower than what was previously installed.
        ii. Example: If firmware version 4 is installed, and then firmware version 0 is installed, the bootloader should only install firmware version 4 or higher.
3. The bootloader may refuse to handle further updates after version 0xFFFF has been installed.

3.3.3.2 Firmware Release Message

Each firmware image has an associated release message that must be installed along with the firmware so it can be printed out at boot time. The release message must be printed over UART if the bootloader agrees to boot the system and should not be printed over UART otherwise (see Section 4.2 for details).

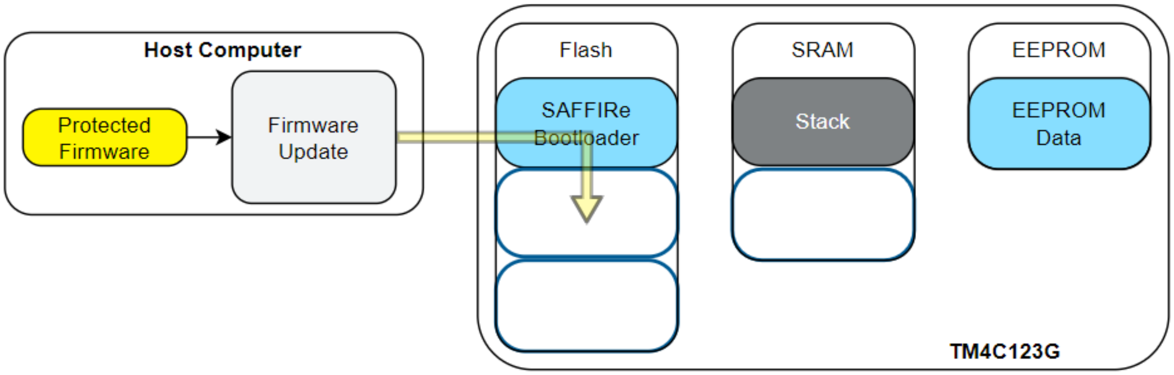


Figure 8 Firmware Update Step

3.3.4 Configuration Load

The host configuration update tool must take a protected configuration image and send it to the bootloader for installation on the device. The tool is only required to perform updates with validly formatted protected configuration images and may return a non-zero exit code if any security violation is detected. This tool does **not** have access to the host secrets file.

The bootloader must accept a new configuration from the host tools and install it to the device such that it can be accessed by the firmware later, including across power cycles. The configuration load routine should only install valid configuration images created by the Configuration Protect host tool. If an update fails mid-way through, the bootloader must continue to operate correctly and meet the functional and security requirements. There are no requirements on how the flight configuration is stored on the device at rest. Unlike Firmware Update, valid protected configurations must be able to be loaded in any order. The bootloader must return to a loop that looks for host commands after performing a configuration load.

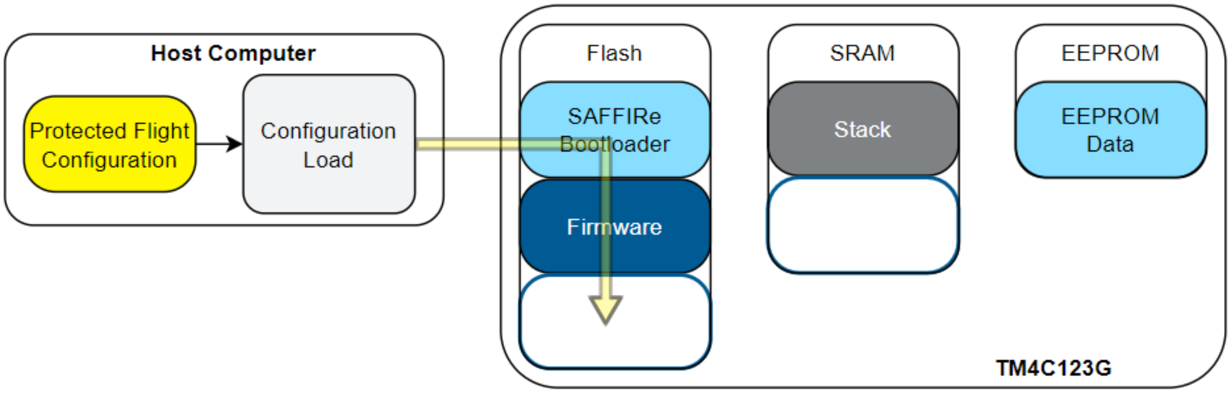


Figure 9 Configuration Load Step

3.3.5 Device Boot

The host device boot tool instructs the bootloader to load the installed firmware and configuration images, print the firmware release message, and begin executing the firmware. If the bootloader detects a security violation in the installed images it may refuse to print the release message. If the device agrees to boot, the boot host tool must save the release message to a file. If the device refuses to boot due to a security violation, the release message should not be saved to a file and the tool may return a non-zero exit code. This tool does **not** have access to the host secrets file. ***Note:*** *This tool is included as a safeguard for the bootloader to refuse to execute untrusted code, which may have security implications during aircraft flight. The aircraft simulation in the emulated environment will check if the boot tool returned a zero exit code and saved the release message to a file before running to prevent accidental usage of the device in-flight. More details on this can be found in the integrity requirements and Attack Phase operation section.*

The bootloader must respond to the Device Boot host tool to load and execute the firmware and configuration installed on the device. Before executing the firmware, the bootloader must send the firmware release message to the host. The bootloader must load the unprotected firmware into a specific address in SRAM. The bootloader must also place the unprotected flight configuration at a specific address in Flash memory. Refer to the Technical Specifications Document for these addresses. If the bootloader detects that the firmware or configuration is corrupted or invalid at any point during boot, it should refuse to boot the firmware and alert the host tools. In this case, the release message should not be sent to the host (see Section 4.2).

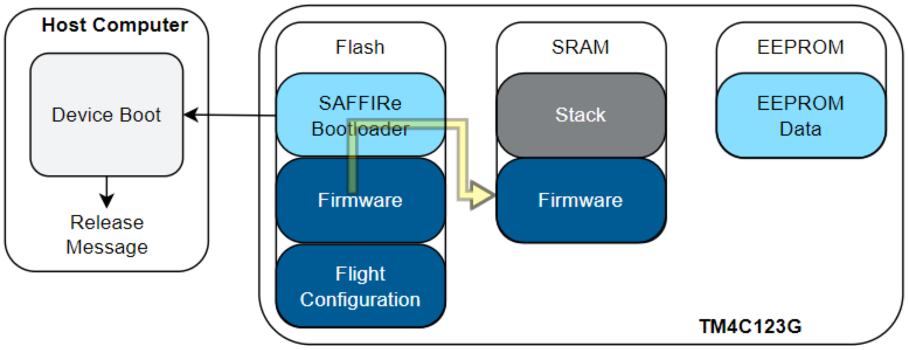


Figure 10 Device Boot Step

3.3.6 Readback

The readback tool requests that the bootloader return data contained in the installed firmware or configuration. The readback tool must indicate to the bootloader which image should be read and how many from that image should be read. The tool may return a non-zero exit code if any security violation is detected. This tool may access the host secrets file.

The bootloader must respond to the Readback host tool to read out unprotected firmware and configuration data. The host readback request consists of a data type (firmware or configuration), and a byte count that indicates how many contiguous bytes to return from the start of the image. The bootloader should only return readback data if an authenticated host makes the request (see Section 4.4 for more details). The bootloader is not required to send back data that is requested outside the firmware and configuration bounds.

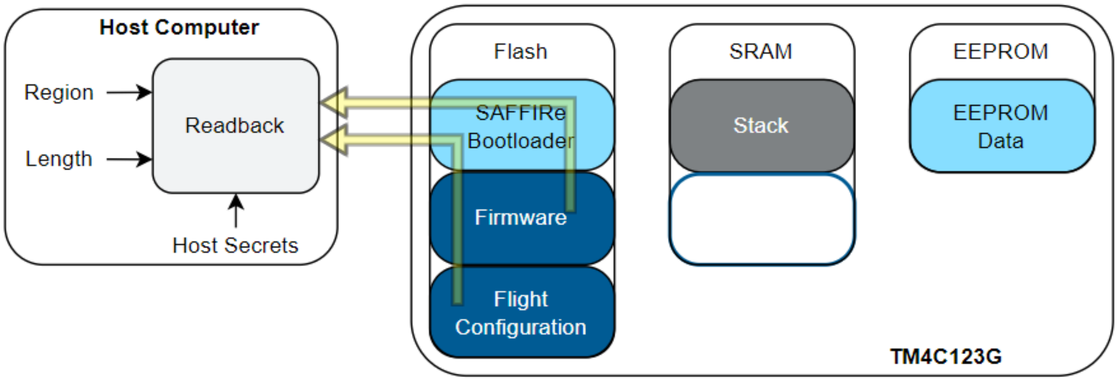


Figure 11 Readback Step

4 Security Requirements

This section defines the security requirements of your SAFFIRe design. These properties will **not** be tested or evaluated during the Handoff phase. Use these requirements to inform your design process, identifying and protecting critical data and code paths.

4.1 Confidentiality

The contents of the firmware and configuration images should never be accessible by an attacker; this includes preventing the attacker from accessing the contents of protected firmware and configuration image files and from extracting this data from the device itself through physical or software tampering. If an attacker can view the raw firmware or configuration data, they may uncover proprietary algorithms and sensitive flight coordinates! The firmware version number and release message do not need to remain confidential. The host secrets file used in the build process will be unavailable to attackers and therefore does not need to be protected by your design.

4.2 Firmware/Configuration Integrity and Authenticity

The SAFFIRe bootloader should only install and boot valid images protected by the secure host tools. If an attacker installs custom or modified firmware and configurations, the flight and the safety of those onboard the aircraft could be compromised. If the bootloader detects that an image was not protected by the secure host tools or was tampered with during an update, it should not install the image. Furthermore, if the bootloader has installed a valid image which has since been modified by any means, the bootloader should refuse to boot the device. The aircraft uses the printing of the release message as the signal that the boot was successful, so the release message should not be printed if the bootloader detects an integrity failure. The integrity of a printed release message does not affect whether the aircraft launches.

4.3 Firmware Versioning

The bootloader should only install firmware images with version numbers newer or equal to the current installed version. If an attacker can install an old version of the firmware that has a bug in it, the system may perform dangerously or reveal critical information. It should not be possible for an attacker to modify the version number of a protected image.

4.4 Readback Authentication

The bootloader should only return installed image data to an authenticated technician that has access to the host secrets. Attackers will not be able to view the traffic between a successful readback, so you do not need to ensure confidentiality over this link.

5 Attack Phase Operation

5.1 Scenario

During the Attack Phase, teams will be assuming the role of a disgruntled ground grew technician who wants to wreak havoc on your company and the safety of the aircraft users. While the technician does not have access to the device while it is being manufactured, teams may submit a hardware trojan for the Flash memory controller that is built into the microcontroller. Once the device has been provisioned with the SAFFIRe bootloader and sent to the depot, the technician will get unlimited physical access where they may use any technique to exploit the design. The technician will also have access to multiple protected firmware and configuration images provided by the secure provisioning facility.

Once the technician has performed some malicious activity, they can signal that the device is ready for flight. At this point, they lose access to the device and the device is put on the aircraft. Before taking off, the aircraft will tell the device to boot. If the bootloader complies and prints the release message, the aircraft will take off. However, if the bootloader refuses due to a firmware or configuration integrity issue or other security violation, the aircraft will stay on the ground and no penalty will be incurred (i.e., no flight-related flags will be dispensed – see the Attack Phase flags).

5.2 Design Deployment

The Attack Phase functionality will be implemented by the eCTF organizers. Deploying a specific design during the Attack Phase will give teams to both a physical and emulated copy of the device, both of which have the **same provisioned bootloader**. This means that any secrets provisioned into the bootloader will be used in both the physical and emulated copies. Each team will be given access to an attack server that makes all attack-phase designs available. The organizers will run the following process to provision an Attack Phase deployment for each design:

1. Create an Attack Phase device emulator with a hardware trojan in the Flash memory controller
2. Build the full system, producing host-tools and device containers
    a. Initial version higher than 1 will be used
3. With the trojan active, load the bootloader into the device emulator and launch the device
4. Protect three firmware images with Firmware Protect
    a. Version 0 – Debug firmware **without** any flags
    b. Version 1 – Firmware with IP Extraction and Firmware Rollback flags inserted
    c. Version 2 – Firmware with IP Extraction flag inserted
    d. More firmware images may be supplied
5. Protect three flight configuration images with Configuration Protect
    a. Configuration 1 – Flight Configuration with Flight extraction flag inserted
    b. Configuration 2 – Flight Configuration with Flight extraction flag inserted
    c. More configuration images may be supplied
6. Run a Firmware Update to install the Version 2 firmware

7. Run a Configuration Load to install Configuration 1
8. Provide all protected images to the attacking team, as well as a copy of the bootloader and EEPROM image protected by MITRE, which may only be installed on the team's keyed device
9. Provide access to the Attack Phase emulator interface for teams to run and interact with the emulated device

6 Scoring

Any flag submitted to the scoreboard will be of the form:

```
ectf{<flag name>_<16 hex characters>}
```

For example, the "Aircraft Crash" flag could look like:

```
ectf{aircraftcrash_0123456789abcdef}
```

6.1 Design Phase Flags

To encourage teams to stay on schedule during the Design Phase, and to give the organizers insight into each team's progress, points will be awarded for reaching certain milestones. Each design-phase flag has a deadline date at which point it can no longer be submitted for points. See the table below for details.

Table 1 Design Phase Flags

Milestone	Description	Target[4] Date	Deadline[5] Date
Read Rules	If you read **all** the rules, you'll know	1/25	2/2
Boot Reference	Provision and boot the example SAFFIRe design to receive a flag (see the README)	1/28	2/9
Use Debugger	Use the GDB target in the top SAFFIRe script to step through a binary and retrieve a flag. See the reference design for details.	1/31	2/16
Design Document	Submit an initial design document containing high-level descriptions of how each host tool and bootloader function will work in your system. It should be clear from your descriptions how your system meets the functional and security requirements. Your design **may** change after submitting this draft, but we recommend updating this as your design changes and including it in your final submission for documentation points.	2/9	2/23
RE Challenge[6]	See Section 6.1.1 below	3/9	4/20
SCA Challenge[6]	See Section 6.1.2 below	3/9	4/20
Bug Bounty	See Section 6.1.3 below	N/A	Handoff [7]

[4] To stay on track with your design, we *highly* recommend trying to hit the Target Dates for each Milestone. However, the flag submission will be accepted up to the Deadline Date.

[5] The exact cutoff will be at 11:59am (eastern time zone) on the deadline date.

[6] The RE and SCA challenges will run the entire length of the competition, but we encourage teams to complete them before handoff as they give advantages for the Attack Phase

[7] Bug Bounties can be submitted up until your team has passed Handoff (notionally, this should be 3/9 but may be later if your submission is later).

6.1.1 Reverse Engineering Challenge

The reverse engineering challenge is back this year with some new goals. Teams will be given binaries of the firmware images running on the navigation computer and bus controller, as well as the source code for the avionics bus "physical layer" interface. Each firmware accepts input over the UART and prints a flag back out over the UART if given the correct input.

Your task is to determine the correct input(s) for each binary, launch them, and send the correct input to receive a flag. Each binary reveals a piece of information that will be useful for creating your own malicious firmware during the Attack Phase that is able to make the aircraft crash (if you can get it to run on a device!) The first binary contains a device ID that you will need, while the physical layer interface code will reveal a vulnerability in the bus logic that allows a device to interfere with aircraft operation. More details on how to run to the binaries and the goals of each reverse engineering challenge component will be released with the binaries.

6.1.2 Side-Channel Analysis Challenge

New this year is the side-channel analysis (SCA) challenge. Teams will collect power side-channel traces from an emulated device running a cryptographic operation and recover the key. The SCA challenge is broken up into three phases: 1. Collect traces containing a cryptographic operation on various inputs; 2. Align and clean up the traces; 3. Recover the key using the aligned and cleaned traces. Upon completing each phase, you will receive a flag that can be submitted to the scoreboard. More details on how to run and complete each phase of the SCA challenge will be released during the Design Phase.

6.1.3 Bug Bounty

If your team happens to find a bug in the reference design, you can earn points for it! Your team will receive 100 points for each bug found, and another 100 points if you submit a corresponding fix. If multiple teams find the same bug, points will be distributed on a first come, first serve basis. Sometimes whether an issue is truly a bug (or a feature!) is a matter of opinion - the eCTF organizers reserve the right to reject bug reports for trivial issues.

6.2 Attack Phase Flags

Each provisioned system in the Attack Phase holds several "flags" that are only revealed if one or more security requirements are compromised. By submitting flags, an attacking team demonstrates that they have compromised security requirements of the target system. For each attack that results in a flag submission, teams must provide a brief description of how they accomplished the attack; failure to do so may result in the revocation of points. The flags are defined in the table below, with detailed descriptions in the following subsections.

Table 2 Attack Phase Flags

Flag Name	Capturing this flag proves that you can compromise…	To Submit this Flag	Requires Aircraft Simulation
Confidentiality Flags			
IP Extraction	Firmware Confidentiality	Extract the flag by reading any protected firmware image	No
Flight Extraction	Configuration Confidentiality	Extract the flag by reading any protected flight configuration	No
Rollback Flags			
Firmware Rollback	Firmware Versioning	Install and boot an old firmware image	No
Integrity Flags			
Data Extraction	Firmware Integrity / Bootloader Execution	Execute unauthorized code that reads data from the device	No
Flight Abort	Firmware and Configuration Integrity	Boot a corrupted firmware or configuration that makes the aircraft deviate from the flight plan	Yes
Aircraft Crash	Firmware Integrity	Boot a malicious firmware that exploits bus vulnerabilities to shutdown aircraft safety features, crashing the aircraft	Yes

6.2.1.1 Confidentiality Flags

Each firmware and configuration image will have a flag built into it before being protected by a design's firmware protect or configuration protect tool. The SAFFIRe confidentiality protections should prevent this flag from being accessible at any point, including when the attacker has access to the protected image and when the attacker has access to a device with an image installed. The firmware will never print out the IP or Flight extraction flags in normal operation, so an attacker will have to bypass the protection features to access the flags.

Example Attack: An attacker is unable to read the contents of a protected image file due to encryption, but a vulnerability in the bootloader allows the attacker to extract the contents of the installed firmware, including the IP extraction flag.

6.2.1.2 Rollback Flag

Attackers will have access to a device that already has firmware with version number 2 installed, so they should be unable to install an older version of the firmware. Attackers will have access to a protected firmware image containing the rollback flag and version number 1. To capture this flag, the provided firmware version 1 image must be installed and booted on the device; extracting the contents of the binary will not be sufficient, as the flag is only created when the firmware is executed on a provisioned device.

Example Attack: A team has a bug in their version number verification logic that allows the installation of an older version number in certain cases. The flag is printed over the UART when the old firmware image is installed and booted

6.2.1.3 Integrity Flags

The safety and security of the aircraft rely on the integrity of the firmware and flight configuration data. The SAFFIRe bootloader should never allow any unauthorized data to be written to the device or unauthorized code to run on the device. Each integrity flag represents a different type of integrity failure with varying levels of impact. Extracting device data requires the attacker to retrieve data from the EEPROM memory. The flight abort flag requires the attacker to modify the behavior of the firmware or contents of the flight configuration to feed unexpected data to the other devices on the avionics bus after takeoff, forcing the pilot to abort the flight. Finally, the aircraft crash flag requires the attacker to install a crafted malicious firmware image that actively tampers with the avionics bus, pushing the aircraft beyond safety bounds and forcing the pilot to eject. Details of each of these flags will be provided to teams on their entry to the Attack Phase.

Example: The bootloader has a faulty integrity check on the Configuration Load, allowing an attacker to load a modified flight configuration to the device. After boot while the device is in-flight, the modified configuration causes bad data to be fed to the avionics bus, which is detected by the pilot who then aborts the flight.

Note: *The Flight Abort and Aircraft Crash flags are only capturable when the aircraft is running, so attackers will not have direct access to the device. Therefore, the device must be prepared such that the attack triggers automatically, likely through modification of the installed firmware or configuration images. When the aircraft simulation detects the conditions for either of these flags, the flags will be dispensed to the team automatically. These flags are only dispensed when using the emulated system.*

6.2.2 Flag Point Values

Since the vulnerabilities to be discovered in the Attack Phase come from other teams' unintentional flaws in the Design Phase, the scoring system is designed to adjust points based on estimated difficulty of capture. More specifically, the point value of any given flag will be adjusted dynamically and automatically based on multiple factors:

- If multiple teams capture the same flag, then the value of that flag will be divided among all the teams that capture it (distribution is not equal – it is weighted based on time of capture to provide more points to earlier captures). Naturally, more difficult attacks will be executed by fewer teams and therefore rewarded with more points.
  Note: *Your total score will drop each time another team captures a flag that you have already captured. This is because the flag points that you were initially awarded need to be redistributed as additional teams capture the same flag.*
- The number of points a flag is worth increases over time as it remains un-captured. This will make the difficult flags more and more appealing as the competition goes on.
- To discourage teams from "hoarding flags" without submitting them, the first capture of each flag will earn the attacker 100 bonus points in addition to retaining the largest share of the flag if also captured by other teams.

6.3 Defensive Points

Defensive points will be automatically awarded over time for each uncaptured flag, beginning once a team successfully completes Handoff. Once another team captures one of your team's flags, that flag will no longer gain additional defensive points.

6.4 Documentation Points

Good documentation will be rewarded to discourage security-by-obscurity. "Good documentation" includes clear and well-commented code, useful descriptions of modules/functions/classes, clear and accurate design documents, and other documents that clearly describe how to read or approach the entire code base.

We are not looking for lengthy documents that describe your implementation in excruciating detail. A concise and clear README.md, including a brief rationale for your security features, combined with well-structured and well-commented code is sufficient for Max points. Quality is valued over quantity.

The maximum number of points that can be scored for documentation is equal to the value of an uncaptured attack flag scored on the last day of the competition, with the actual amount being a percentage of that maximum:

- **Max** - Exemplary documentation, comments, and code structure; clear and easy to understand
- **75%** - Good comments and high-level documentation
- **50% -** Good comments, but lack of clear high-level documentation
- **25%** - Confusing code and little or no actual documentation
- **0%** - Confusing or deceptive comments and documentation

Points for documentation will not be awarded until the end of the Attack Phase. Honest feedback on documentation from other teams will be solicited and factored into the final point determination.

6.5 Write-Ups

There will be an opportunity after the Attack Phase for teams to provide a two-page write-up for additional points. Further details on the content and format of the write-ups will be provided during the Attack Phase.

7 Rules

Most rules are described and explained throughout the challenge description in the earlier sections – please read this entire document! This section is intended as a concise summary of the most important rules.

(1) **You may not make any attempt to gain additional privileges on any servers used for the eCTF, probe the system or network, expose any ports, or use them for any other purpose beyond what is explicitly in-scope for the eCTF competition.**

- **Usage of these systems will be monitored, and misuse of the provided servers may result in expulsion and disqualification from the competition, and/or legal action.**
- **Only student-designed systems that are explicitly designated as targets are within scope for attacks, and such attacks are only allowed during the Attack Phase.**

- **If you've read all the rules then you can prove it with a SHA1 hash of this pdf file.**
- **If you are unsure if certain actions are within the scope of the competition, contact the eCTF organizers first.**

(2) In addition to the rules provided by MITRE, participants should also adhere to all the policies and procedures stipulated by their local organization/university.

(3) MITRE reserves the right to update, modify, or clarify the rules and requirements of the competition at any time, if deemed necessary by the eCTF organizers.

(4) When submitting your secure design, all source code and documentation must be shared.

- This is to discourage security-by-obscurity, as well as to accelerate attack development and encourage more sophisticated techniques for both sides.
- Creating any part of your submission in an obfuscated manner or using an esoteric programming language is considered security-by-obscurity and is not allowed. Please contact the organizers if you have any questions about this.

(5) You may only attack the student-designed systems explicitly designated as targets, and such attacks may only occur when you are in the "Attack Phase" of the competition.

- Attack deployments will be running on servers that are part of competition infrastructure – these servers are NOT in scope for attack.
- A MITRE bootloader will be running on the physical device that securely loads attack-phase designs – this bootloader is NOT in scope for attack.
- If you have any question about whether a component is "in scope", please reach out to the organizers for clarification.

(6) All flags must be validated by submitting a brief description of the attack.

- Attack descriptions should be sufficiently detailed to allow the defender to correct their vulnerability. eCTF admins may invalidate points for flags that are not validated before the completion of the eCTF.

(7) Flag sharing across teams is not permitted and will result in immediate disqualification.

(8) No permanent lock-outs are allowed. See the Technical Specifications Document for timing and performance requirements.

(9) Your system must work with the emulation system that was provided. Switching to a different platform or modifying the emulation backend during the Design Phase is not allowed.

(10) All documents that are submitted (e.g., design document and write-ups) must be in PDF format.

8 Frequently Asked Questions

Is it OK to obfuscate our source code to make it more challenging to understand and attack?

No. Obfuscations performed at compile-time (e.g., to make binary reversing more challenging) are OK, but your source code needs to be written in a clear and maintainable fashion. It should be well-commented and/or otherwise documented clearly.

Can we add intentional delays during boot to make it more difficult for an attacker to collect large numbers of observations?

There should not be any intentional delays that could push performance past the timing requirements in the Technical Specifications Document.

If your system detects that it is under attack, additional delays are OK, but must be limited to no more than 5 seconds. Permanent lock-outs or self-destruction is not allowed.

Can we attack another teams' development environment?

No! Everything other than the provisioned devices, the host tools, and the firmware and configuration images are considered out-of-bounds. In other words, there is **nothing** that you can attack until your team enters the Attack Phase.

Is social engineering in-scope for this competition? Can we send phishing communications to other teams to trick them into revealing their secrets?

No, please don't do this. Keep your attacks technical. We love creative ideas, but this one can easily violate state and federal regulations.

Can we submit the reference design or a design with trivially-defeated security so we can move into the Attack Phase?

No. As this is a design-build-attack-style competition, teams must submit a design that exhibits significant effort on the design and build components. It is up to the discretion of the eCTF organizers as to what level of modification counts as "significant effort", so please contact the organizers before submitting an extremely pared down version of your design. ***Note:*** *you may submit designs up to the last day of the competition – and there have been very successful teams that did not submit on time – so don't panic if your design isn't ready on the first day of the Handoff Phase.*

Can we attack MITRE infrastructure, either files that have been protected by MITRE for secure distribution of provisioned systems, or the development and attack servers?

No. Any infrastructure that has been created by MITRE is off-limits for this competition. The eCTF organizers put these capabilities in place to make the competition smoother for everyone and should be considered transparent when attacking provisioned designs. ***Note:*** *when submitting Attack Phase flags to the scoreboard, the attacking team must submit a brief summary of how the flag was captured to the eCTF organizers. If capturing a flag involved any tampering with MITRE infrastructure the flag points will not be awarded.*

What is a "provisioned system"?

A provisioned system is a set of files and devices provided by the eCTF organizers for use during the Attack Phase. In this competition, the provisioned system includes a physical or emulated microcontroller running the design's SAFFIRe bootloader and firmware and configuration images protected by the design's protect tool using the build artifacts obtained from building the system. The secrets files will **not** be distributed.

9 Changelog

- V1.1:
    - Add clarification that loading the SAFFIRe bootloader into the emulated device is in scope for hardware trojan attacks
    - Add rules that the host tools that interact with the bootloader may return non-zero exits due to security violations
    - Add load step to the Attack Phase provisioning flow with the trojan enabled
    - Update cover flag logo