Design and Implementation of an Inline Security Gateway and State-Aware Validation Filter for Legacy MIL-STD-1553 Avionic Buses

Jackson McCullough

July 2026

1 Research & Preliminaries

Before writing any code or simulating traffic, the project required a deep learning phase to understand MIL-STD-1553 bus architectures and communication standards. The following foundational documents were reviewed to guide the system's design:

- *"MIL-STD-1553 Tutorial and Reference"* – Alta Data Technologies
- *"Digital Time Division Command/Response Multiplex Data Bus"* – DoD Interface Standard
- *"MIL-STD-1553 Programmer's Guide (C/C++)"* – AIM Avionics Databus Solutions

2 System Architecture & Tech Stack

To create an accurate hardware-in-the-loop simulation, the project required a structured approach to generating traffic and modeling the interactions between a Bus Controller (BC) and a Remote Terminal (RT).

The core technologies used in this project include **Python, C, an STM32 Nucleo board, and the UART protocol**. The Nucleo board serves as the microcontroller for traffic simulation, utilizing UART for hardware communication. The development was divided into two primary phases:

- **Phase 1:** Environment Setup & Protocol Framing
- **Phase 2:** Simulation & Attack Vector Testing via Secure Gateway Hardware

3 Phase 1: Environment Setup & Firmware Development

3.1 Python Simulation Scripts

To emulate the avionics network on a host machine, two primary Python scripts were developed. The first, `bus_simulator.py`, establishes the data framing rules required by the 1553 standard, structuring command words, status words, and data words. To defend against replay attacks, a custom 5-bit sequence tag was added to the data words. While this 5-bit sequence is **not** part of the official MIL-STD, it is a project-specific implementation demonstrating a modern method for validating data uniqueness.

The second script, `serial_bus_tester.py`, acts as the hardware-in-the-loop orchestrator. It runs the full sequence (Bus Controller $\longrightarrow$ Remote Terminal $\longrightarrow$ Gateway) and evaluates the gateway's responses. The gateway either accepts the data word or rejects it with a **0xFFFF** flag. This script is crucial for testing sequence validation and timing windows. If a data word arrives too late, it is dropped. Repeated test cases confirmed that failed attack attempts do not desynchronize the gateway's internal sequence counter.

3.2 STM32 Microcontroller Firmware (C)

The security logic was deployed to the STM32 Nucleo board using custom C firmware. (Note: Header files were created for each script but are omitted here for brevity).

The core filtering logic is housed in `secureGateway.c`. Rather than generating test traffic itself, it sits on the UART line and passively monitors traffic, deciding in real time whether to relay or reject it. It operates as a two-state machine:

1. **Idle State:** Watches for and relays command words addressed to RT 5.
2. **Response-Waiting State:** Validates the subsequent data word based on two independent conditions: the embedded sequence ID and a strict timing window (rejecting responses that are too slow or suspiciously fast).

If both checks pass, the word is forwarded unchanged. If either fails, the board lights an onboard LED to indicate a dropped packet and transmits a `0xFFFF` sentinel back over UART. This is tracked using an 8-bit counter, `uint8_t expectedSeqID = 0`, which only increments after a fully valid exchange. This ensures malicious packets do not disrupt synchronization.

The infrastructure is handled by `main.c`. It initializes the Hardware Abstraction Layer (HAL), configures the system clock (via the internal HSI oscillator), and sets up the USART2 peripheral for 115200 baud communication over the ST-LINK virtual COM port. It also manages the SysTick interrupt handler, providing the clock source that `secureGateway.c` uses to measure response latency.

4 Phase 2: Simulation & Hardware Testing

With the firmware successfully flashed to the microcontroller, a baseline run was executed using `bus_simulator.py` to verify standard traffic flow. The simulation generated arbi-

trary baseline data (e.g., an altitude of 10,000 feet) to confirm the gateway was correctly parsing and accepting valid data words.

```
(.venv) PS C:\Users\jacks\PycharmProjects\1553_proj> python .\bus_simulator.py
--MIL-STD FRAMING TEST--

 Command Word: 0x2C21 , Binary Word: 0010110000100001
 -> Target RT: 5 | T/R: Transmit | Subaddresses: 1 | Wrod Count: 1

 Data Word: 0x2710 , Binary Word: 0010011100010000
 -> Payload Telemetry (Altitude): 10,000 ft

 Status Word: 0x2800 , Binary Word: 0010100000000000
 -> Responding RT: 5 | Status: Nominal (No Errors)
```

Figure 1: Terminal output showing successful baseline traffic execution.

During this baseline phase, the Nucleo board remained in its normal operational state, seamlessly passing clean traffic without triggering any rejection flags.

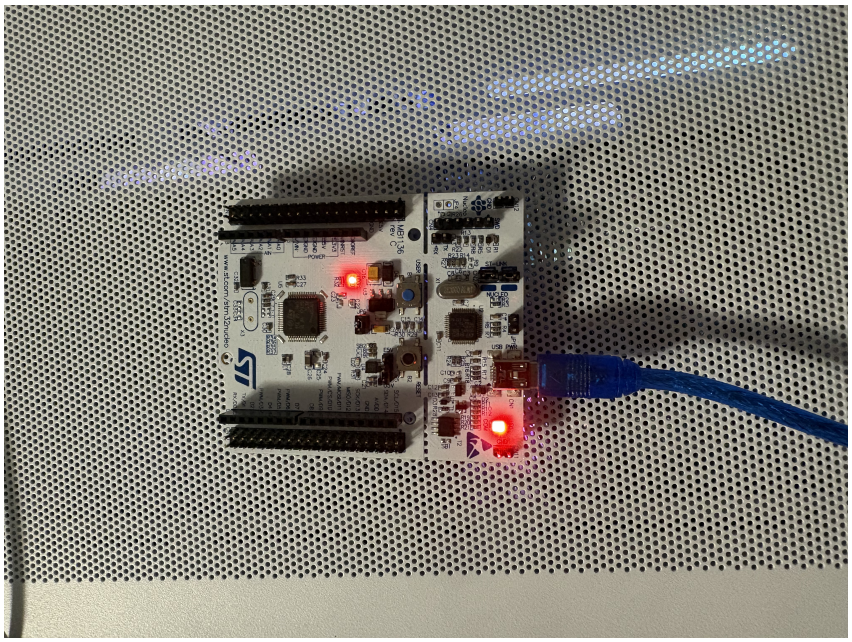

Figure 2: Nucleo board observing basic traffic with no dropped packets.

4.1 Executing Attack Vectors

Once the baseline was established, `serial_bus_tester.py` was used to introduce malicious traffic, specifically targeting the system with replay attacks and timing delays. The console output clearly demonstrated the gateway successfully intercepting the anomalies:

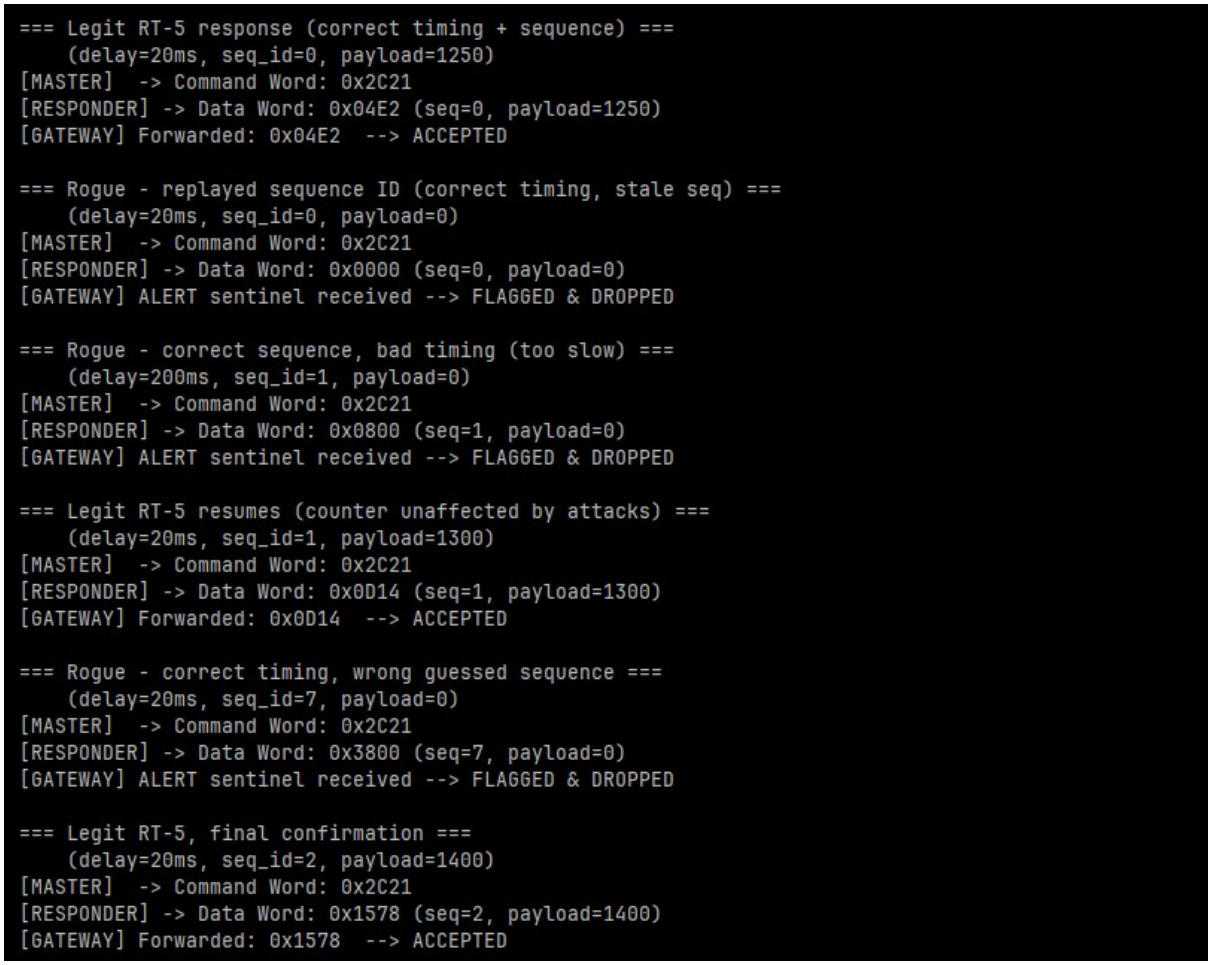

Figure 3: Console output displaying flagged transactions and 0xFFFF rejections.

This rejection was physically verifiable on the hardware. The microcontroller was programmed to flash a green LED whenever a transaction was flagged and dropped. During the attack simulation, the board successfully illuminated the LED, providing visual confirmation that the state-aware filter was working in real time.

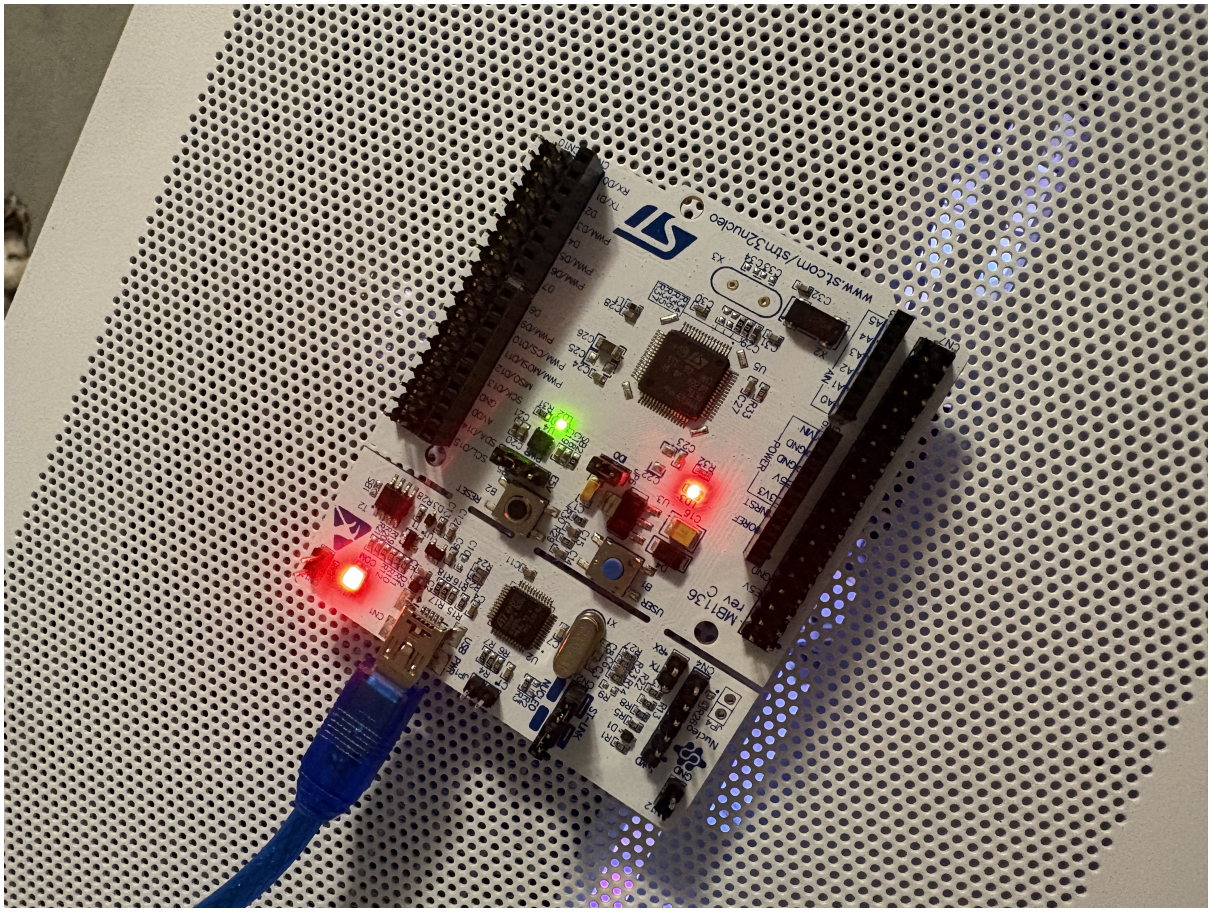

Figure 4: Hardware LED illuminating to indicate dropped malicious traffic.

This confirms that malicious traffic was received, flagged, and dropped by the microcontroller, while allowing clean traffic to proceed with no complications.

5 Conclusion

This project successfully demonstrated the viability of implementing an inline, state-aware security gateway to protect legacy MIL-STD-1553 avionics buses from modern network exploits. By introducing a custom 5-bit sequence tag and enforcing strict latency windows, the STM32-based gateway effectively identified and neutralized both replay and timing attacks in real time. The hardware-in-the-loop simulation proved that malicious data words could be instantly flagged (with `0xFFFF`) and dropped without desynchronizing the internal state or interrupting the flow of legitimate avionics traffic. Ultimately, this serves as a strong proof-of-concept for how legacy military and aerospace communication protocols can be retrofitted with lightweight, highly effective inline filtering to ensure data integrity and operational security.