The Compromised Satellite Peripheral Dilemma

Rachel McAmis *†, Connor Willison *, Richard Skowyra *, Samuel Mergendahl*

* *MIT Lincoln Laboratory*
Secure Resilient Systems & Technology
Lexington, MA
{rachel.mcamis, connor.willison, richard.skowyra, samuel.mergendahl}@ll.mit.edu

† *University of Washington*
Seattle, WA
rcmcamis@cs.washington.edu

***Abstract*—Satellite systems enable many capabilities for their users, such as high-speed, low-latency communications, weather forecasting, geographic imaging, and defense applications. As customers increase their reliance on this critical infrastructure, the risk of attack only increases, particularly from highly-resourced adversaries. However, in this work, we demonstrate that common existing space system software platforms are poorly equipped to handle malicious satellite peripherals. Using NASA's popular open source core Flight System software (cFS), we show that with current satellite software and industry-standard reliability techniques, a system designer will inevitably confront a dilemma: Either the system deploys countermeasures against malicious components and suffers degraded nominal performance, or the system cannot survive malicious components. We conclude by proposing challenges and considerations towards resolving this dilemma.**

I. Introduction

An unprecedented number of commercial satellite launches [1] has recently led to the emergence of space systems as an unofficial category of critical infrastructure. These commercial systems provide high-speed, low-latency communications, position, navigation, and timing (PNT) [2], weather forecasting [3], and geospatial imagery [4], among other applications. Commercial satellites deploy most frequently into Low Earth Orbit (LEO) due to lower launch costs [5] as well as the desire to use commercial-off-the-shelf (COTS) components, which are less suited to the longer mission lifespans and harsher conditions of higher orbits [6].

Given the critical nature of space systems, cybersecurity is of utmost importance. Satellite security has traditionally focused on the protection of communication links [7], but more expansive threat models have been proposed recently [8], [9]. For example, the threat of memory corruption should concern satellite designers since attackers can leverage these common vulnerabilities to launch advanced code reuse attacks and circumvent many defenses.

One understudied aspect of satellite system security is the inherent trust of the micro-controller software found in satellite peripheral subsystems. As space companies increasingly rely on outsourcing components to decrease cost and development time, the risk of software supply-chain attacks originating from peripheral subsystems also increases. Common satellite peripherals include star trackers, reaction wheels, or other sensors and actuators, and the software running in these peripheral subsystems is often proprietary and treated as a black-box by mission designers. This can lead to situations where the flight computer places undue trust in the code and data of the peripheral subsystems. While some satellite security work has touched on the risk of software supply chain attacks on satellites [8], as well as the risk of external physical disruption of satellite sensors [9], the impact of a compromised peripheral subsystem on the flight computer has not been experimentally evaluated. At the same time, adversary incentives for such attacks have increased due to emergence satellite fleet and bus companies that produce at-scale. A supply chain attack on satellites could now impact an entire fleet at once.

In this work, we explore the impact of a compromised peripheral subsystem on a satellite. We investigate a typical Guidance, Navigation, and Control (GNC) subsystem on a spacecraft that uses a star tracker sensor peripheral, an Attitude Determination and Control (ADCS) feedback loop, and reaction wheels as actuator peripherals. We demonstrate that without any protections, a malicious star tracker peripheral can cause GNC failure through easily executed attacks. Specifically, we demonstrate that the ADCS software component on the on-board computer acts erroneously when the sensor peripheral either sends *corrupted data* or nominal data at *unexpected frequencies*.

Additionally, we show that two commonly deployed software resiliency strategies are insufficient against a malicious peripheral. Namely, we study the detection of anomalous

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Department of the Air Force under Air Force Contract No. FA8702-15-D-0001 or FA8702-25-D-B002. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Air Force. © 2026 Massachusetts Institute of Technology. Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

Workshop on Security of Space and Satellite Systems (SpaceSec) 2026
23 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-1-1
https://dx.doi.org/10.14722/spacesec.2026.23045
www.ndss-symposium.org

sensor input against configurable thresholds, and secondly, the use of sensor fusion with a redundant set of sensor peripherals. Historically, these strategies are believed to provide resilience against faulty components. However, we find that neither strategy is effective against *malicious* components.

We observe a design trade-off for system configuration on satellites given these resiliency strategies: Either the system deploys countermeasures against malicious peripherals and pays a cost in nominal mission performance, or the system cannot survive against malicious peripherals. This dilemma suggests that satellite operators face significant challenges if they use existing tools to defend against malicious peripherals.

Previous work has mentioned the dilemma of attack detection sensitivity and its impacts on reliability and runtime [10], but this problem has yet to be considered in the satellite context. Lu et al. highlights the need for more domain-specific work on cyber-physical system attacks and recovery research, as each domain has its own constraints and challenges [11], and we aim to address this gap by providing evaluation of possible attack scenarios in the satellite domain. The satellite domain poses different constraints compared to other robotic vehicles like drones, a common platform for testing sensor attacks and associated recovery techniques [12]–[14]. Compared to drones, satellites must handle a much longer mission, taking up to multiple years. Satellites must be able to handle very stealthy attacks that may only cause measurable impacts over periods longer than a single drone mission. A satellite's location must be precise to stay in the correct orbit while pointing its communications module and sensors in the correct direction, whereas a drone might be able to land many meters away from its target landing destination while still executing a safe mission. Other unique considerations for satellites are their possible vulnerability to hardware faults due to radiation in orbit, and the complexity of multiple control systems and sensors operating on a machine with limited power and computation to put towards attack detection.

After we provide the relevant background necessary in Section II and introduce our studied threat model in Section III, our work makes several contributions:

- We introduce a series of malicious behaviors that a compromised peripheral can employ as an attempt to degrade a satellite mission in Section IV. We show these for the first time on a satellite system within the emulated system described in Section V.
- We characterize the impact of a malicious satellite peripheral micro-controller on the on-board flight computer in Section VI.
- Further, in Section VI, we implement and evaluate two resilience strategies commonly used in satellite missions and highlight their limitations.
- We argue that existing anomaly detection and recovery systems are insufficient for the satellite context in our attack model.

II. Background

a) Satellite System Architecture: Satellite systems typically consist of a central flight computer that interfaces with several peripherals. The flight computer orchestrates communication between each of the peripheral subsystems, hosts software to perform *soft* real-time control loops, and performs command and data handling tasks. Peripherals may have responsibilities ranging from non-volatile storage to mission-specific sensors and actuators. As separate hardware components connected to the on-board computer over a serial link, these peripherals contain their own microprocessors which manage the low-level, *hard* real-time control loops specific to that peripheral. Many peripherals on a satellite interface to sensors and actuators to enable the flight software on the central flight computer to interact with the physical environment.

b) Guidance, Navigation, and Control: Star trackers are sensors which are employed on satellite buses to gather information about the spacecraft's attitude (orientation). A star tracker is an optical sensor which is purpose-built for capturing starfield images and computing attitude solutions using a database of known celestial bodies. The result of a star tracker measurement is an attitude quaternion describing the orientation of the spacecraft. This is used to inform an attitude estimate as part of a sensor fusion strategy, which combines data from one or more star trackers and other sensors to reject noise and achieve higher precision. Moreover, reaction wheels are often deployed to adjust the satellite orientation as needed. See Figure 1 for a high-level view of how a star tracker might work, with potentially malicious actors in the sensor data pipeline.

c) NASA core Flight System: The core Flight System (cFS) [15] is a modular flight software framework developed by NASA. cFS is written in C and uses a message passing architecture over a software bus to facilitate communication between applications. Each application in cFS is separately compiled and runs in a dedicated thread. The modular design of cFS contrasts sharply with traditionally designed flight software systems, which typically have monolithic architectures. The modular nature of cFS benefits mission reusability.

The limit checker is an optional cFS application that monitors software bus telemetry for anomalous values according to a configurable set of *watchpoints* and *action points* [16]. The watchpoints define which application messages (and their associated values) to monitor. The action points define what conditions indicate anomalous values and specify the relevant remedial actions to take under any met conditions, such as ignoring future packets from the anomalous source.

d) Satellite Security Research: There exist many challenges to implementing security in the satellite context, including the inability to physically access the satellite post-launch, emphasis on safety and availability over confidentiality, and handling radiation faults [8].

Recent work has shown that satellites in industry and academia practice insufficient security [17], [18]. Jero et al. propose future approaches to a more in-depth security posture beyond communication security, such as reverting to a “safe mode” [8]. However, safe mode typically does not prevent malicious peripheral attacks, as it may inherently trust important sensors like the star tracker.

While previous satellite security research has touched on supply chain security [8], [19], [20], to our knowledge none have evaluated the details of what such a supply chain attack entails, including whether firmware or software is compromised, how and when the satellite might be compromised, or the persistence of such a compromise. For example, Pavur et al. list a “threat matrix” toolbox through synthesis of previous satellite security research [19]. This toolbox does not include any mention of malicious peripherals or the impact of incorrect sensor data through stealthy attacks. However, Pavur et al. does highlight institutional barriers leading to the fact that “almost no technical research exists on the defense and monitoring of systems in orbit”, and that this is likely “due to substantial logistical barriers involved in accessing and conducting research with representative space hardware” [19]. We overcome this barrier through extensive effort in building out a representative satellite simulation using NASA cFS along with configurations for different peripheral attacks and detection methods.

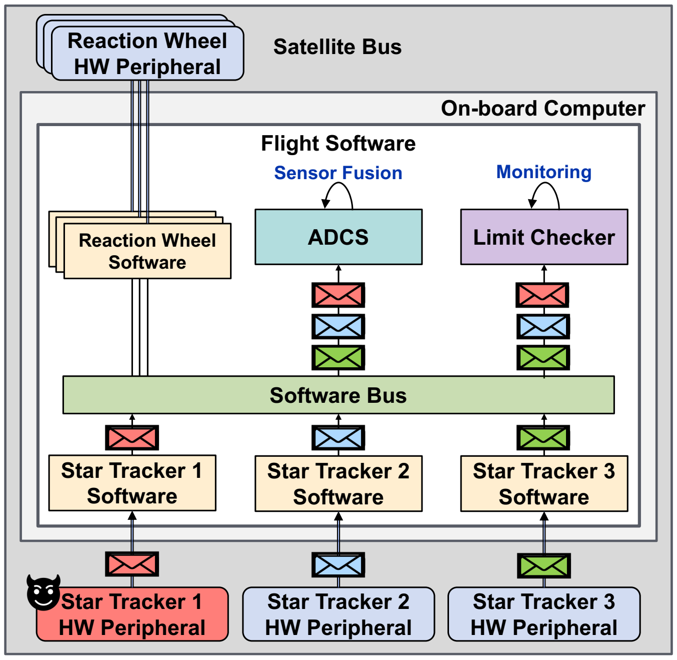


Fig. 1: Threat model of malicious peripheral.

III. Threat Model

We assume a typical satellite system architecture in which an on-board computer deploys flight software such as cFS in order to manage multiple peripherals connected to the on-board computer over a serial link. Specifically, there exists three COTS star trackers and multiple reaction wheel hardware peripherals each with their own micro-controllers connected to the on-board computer that operates the Attitude Determination and Control (ADCS) soft real-time loop for Guidance, Navigation, and Control (GNC). To improve the reliability of the system, the ADCS component performs sensor fusion on the redundant star tracker data, and a limit checker application monitors the star tracker data for signs of anomalies. Additionally, we assume one star tracker to be compromised. This could be due to a malicious supply chain attack given the increasing reliance on blackbox COTS components, or some other external force such as sensor spoofing [9]. From the compromised star tracker peripheral, the adversary attempts to degrade the overall attitude of the satellite. See Figure 1 for more details.

In a compromised supply chain scenario—which is the main threat scenario for this paper—the peripheral has access to the true sensor measurements. Because of its malicious firmware onboard, the peripheral can construct its measurements based on knowledge of the true sensor values. This enables a finer-grained attack that differs from an honest but faulty setting, where the sensor values or timing at which values are sent are not intentionally constructed.

IV. Attacks and Common Mitigations Overview

In this section, we introduce two types of attacks that a compromised satellite peripheral could launch, as well as two resiliency strategies commonly deployed in space system software. As mentioned in Section III, we contextualize these attacks and defenses with respect to the GNC subsystem of a satellite.

a) Attack Impact: In a real satellite mission, incorrect attitude data can cause the satellite to be incorrectly positioned, thereby moving the payload and communications module away from where they should be relative to Earth. This can ruin the mission by preventing collection of the correct data (e.g., if the payload is a sensor collecting images), preventing the satellite from pointing its solar panels properly to the sun and thus losing power, or causing the satellite to lose communication with its ground station due to incorrect radio positioning. Incorrect sensor data can not only cause the satellite to point in the wrong direction, but can also cause the satellite to tumble (i.e., spin uncontrollably).

b) Studied Attacks: We separate our attacks into two axes: attacks related to manipulating time without any malicious data manipulations, and attacks sending incorrect data without any malicious time manipulations. The first attack we evaluate is a **Manipulated Data Attack**, where the compromised star tracker peripheral sends incorrect data values. The second attack we evaluate is a **Manipulated Time Attack**, where the star tracker data is correct, but sent at an unexpected frequency. Specifically, in each time period, we withhold the N star tracker data points and instead send them as close together as possible in a burst of size N. Of course, an adversary could combine these two attack techniques (e.g., flooding maliciously crafted data), but we study the disjoint version of the attacks in order to ensure one technique does not influence the impact of the other technique.

c) Satellite Reliability Techniques: When a satellite developer is preparing for a mission, they often value reliability over security [17]. In particular, the nominal behavior of a satellite must work extremely reliably, but developers must also prepare for issues such as radiation-induced faults. To survive faults and increase overall reliability, satellite developers often use multiple approaches. The first common reliability approach on satellites is to use redundancy. For example, a GNC subsystem might deploy three different sensors each separately determining orientation, and fuse together measurements from all three sensors to output a more robust estimate of the true orientation of the satellite. Indeed, in our evaluation (Section V), we simulate data from three star trackers, where only one is malicious. While redundancy is useful for recovering from faults in the nominal case, using redundancy when at least one sensor is malicious can be ineffective, as shown later in Section VI.

The second reliability approach commonly used is a threshold-based limit checking scheme to monitor sensor data for analogous conditions. For example, the NASA cFS flight software contains the Limit Checker application for this purpose. The limit checker is used to detect when values are out of range and trigger action on those detections. For example, the limit checker can have a watchpoint that detects when the temperature of a satellite's battery is too high, and can trigger an action to cool the satellite. Or, a watchpoint could verify that the satellite's star tracker data is within the expected range for its unit of measurement (between -1 and 1 for quaternions), and shutdown the sensor if that range is not met, with the assumption that a temporary fault may lead to an out-of-range value. While useful in honest settings, we show in our results that a limit checker is poorly suited for handling malicious peripherals.

V. Experimental Setup

In this section, we describe our satellite emulation environment and test flow for launching the attacks introduced in Section IV.

A. Emulation framework

We emulate a typical GNC subsystem using a software-only test harness that deploys cFS on Linux as the flight software for the on-board computer. The spacecraft is instantiated within the Basilisk astrodynamics framework [21], actively used in research and industry for testing, which simulates the rigid body dynamics of the vehicle in Low Earth Orbit (LEO). Additionally, cFS deploys custom applications from Basilisk that implement a simple attitude control loop using emulated sensors and actuators to create a closed-loop emulation of the satellite.

In particular, a star tracker application ingests attitude state information from the simulation to model the behavior of a physical star tracker sensor. The sensor data is forwarded to the ADCS application, which uses an attitude control algorithm from Basilisk to compute the reaction wheel torques necessary to maintain an attitude reference. A reaction wheel application consumes the torques and forwards them to the simulation, where they are integrated into the rigid body dynamics to emulate the behavior of physical reaction wheels. The simulation feedback loop through cFS executes at a rate of 1 Hz and achieves stabilization to a given reference attitude within 30 seconds under honest (nominal) conditions. This simple attitude control test harness allows for the execution of maneuvers by sending a command to change the attitude reference.

Because we are more interested in the impact of a compromised peripheral—rather than how it became compromised—the test harness also includes commands to trigger attacks from the malicious star tracker, as well as system configurations to adjust the settings for the deployed resiliency mitigations.

B. Attack Setup

The procedure is as follows for both manipulated time and manipulated data attacks:

1) *Step 1: Ensuring limit checker metrics meet honest sensor requirements.* Before performing any attacks, we first define the limit checker configuration based on the distribution of expected nominal data and the level of acceptable risk for the mission (e.g., tighter min and max limit thresholds are more conservative against faults, but may cause degradation of nominal performance). This procedure represents the actions a satellite developer might take to improve the satellite's reliability. We only tune the limit checker for star tracker data so as to only focus on one satellite subsystem.
2) *Step 2: Perform attacks.* We then perform attacks that 1) operate within these limit checker thresholds so as not to trigger any action, and 2) operate slightly out of those thresholds to show the impact of detecting an attack that is not stealthy. We show that operating within these thresholds can still lead to mission failure, and even detection of non-stealthy attacks has performance slowdowns.

We evaluate these limit checker settings and attacks both with and without sensor redundancy, and show that redundancy cannot necessarily help survive attacks, even if only one out of three star tracker sensors are malicious.

C. Testing

The test flow is as follows. First, a maneuver is triggered by an attitude change command to reach a target attitude after starting from a different initial attitude. Then, after a time delay, a command triggers the attack. If the satellite has not achieved the reference attitude to a given tolerance after 120 seconds, the simulation is terminated. In particular, our tests attempt an attitude maneuver from starting quaternion values of

$$q_0 = [1, 0, 0, 0]^T$$

to the target quaternion values of

$$q_1 = [0.6002842, 0.4617571, -0.4617571, -0.4617571]^T$$

These target orientation were chosen without loss of generality and at random.

VI. Attack Results

In this section, we leverage the emulation framework from Section V to report the impact of the data manipulation and timing attacks. Additionally, we report how typical resiliency defenses will affect honest (nominal) and malicious (attack) mission scenarios. Results are averaged over 5 trials for all tables, where each entry of the table is the area under the curve of the attitude error metric described below. A smaller value in a table entry means a more successful defense as it represents less cumulative attitude error. The reader can refer to the appendix for example-based visualizations of attack impacts.

A. Attitude Error Metric

Attitude error is defined as the angular displacement in radians of the satellite's $SO(3)$ orientation relative to the goal orientation, q_1. Error is computed by finding the magnitude of the axis-angle representation [22] of the quaternion quotient between current and goal orientation, as follows:

$$\Delta q(t) = q(t)q_1^{-1}$$
$$\Theta(t) = 2\text{acos}(\Delta q_w(t))$$

Where:

$$\begin{aligned} q(t) &: \text{Quaternion attitude time series} \\ q_1 &: \text{Goal orientation} \\ \Delta q(t) &: \text{Orientation relative to goal} \\ \Theta(t) &: \text{Angular displacement (radians)} \end{aligned}$$

The attitude error is taken as $\Theta(t)$, the angular displacement relative to goal. For coarse-grained evaluation of three-axis stabilization performance under various attack scenarios, an additional area-under-the-curve (AUC) metric is defined as $\Theta(t)$ integrated through time:

$$\text{AUC} = \int_{t_0}^{T} \Theta(t)dt$$

Where t_0 is the fixed time offset into the maneuver at which the cyber attack is launched. T is a predefined timeout value at which the satellite is known to achieve its goal orientation under nominal conditions, with no cyber attack taking place. AUC is computed using rectangular Euler integration for each simulated scenario. It is assumed that AUC takes on some ideal, minimal value under nominal conditions when the satellite converges smoothly to the goal orientation. Given that $\Theta(t)$ is non-negative by definition, any sub-optimal deviation from the expected trajectory manifests as an increase to AUC. Therefore, AUC may be taken as a metric for evaluating the degree of disruption to the three-axis stabilization in the presence of a cyber attack. We use the AUC measurement in Tables I through X.

TABLE I: Honest case: turn star tracker off if quaternion difference is *smaller* than δ_{min} for n consecutive times. NO redundancy defenses

	Consecutive limit checker alarms before response				
δ_{min}	2	4	6	8	10
0.00001	5.127	5.127	5.126	5.127	5.127
0.0001	5.126	5.127	5.127	5.127	5.127
0.001	5.127	5.127	5.127	5.127	5.127
0.01	118.369	5.127	5.127	5.127	5.126
0.1	168.598	118.369	5.127	5.127	5.127

TABLE II: Honest case: turn star tracker off if quaternion difference is *smaller* than δ_{min} for n consecutive times. WITH redundancy defenses

	Consecutive limit checker alarms before response				
δ_{min}	2	4	6	8	10
0.00001	5.133	5.132	5.132	5.132	5.133
0.0001	5.133	5.133	5.132	5.132	5.132
0.001	5.132	5.133	5.133	5.133	5.132
0.01	117.794	5.133	5.132	5.132	5.133
0.1	168.677	117.801	5.133	5.132	5.133

B. Data Manipulation Attack Results

We evaluate the impact of malicious sensor values on the attitude of the satellite. We study two defense strategies against this attack: first, redundancy of three star trackers (where only one is malicious), and second a defense of detecting when the variation between consecutive quaternion values is too high or low. In this section, we primarily study how an operator sets and applies limit checks to flag any unsafe *variation* of quaternion values. Each test also applies a limit check to ensure the raw quaternion values are sensible (i.e., any quaternion values outside the range of $[-1, 1]$ are flagged). These are reasonable thresholds to set on a mission, as nonsensical attitude values or dramatic attitude shifts both could damage the satellite and represent faulty behavior.

1) Determining Thresholds for Honest Case: We first run functional tests on different possible limit checker configurations and redundancy scenarios when the star tracker(s) are

TABLE III: Honest case: turn star tracker off if quaternion difference is *bigger* than δ_{max} for n consecutive times. NO redundancy defenses

	Consecutive limit checker alarms before response				
δ_{max}	2	4	6	8	10
0.00001	168.598	118.368	5.127	5.127	5.127
0.0001	168.599	118.370	5.127	5.127	5.127
0.001	168.595	118.368	5.127	5.127	5.127
0.01	168.592	5.126	5.127	5.127	5.127
0.1	5.127	5.127	5.126	5.126	5.127

TABLE IV: Honest case: turn star tracker off if quaternion difference is *bigger* than δ_{max} for n consecutive times. WITH redundancy defenses

	Consecutive limit checker alarms before response				
δ_{max}	2	4	6	8	10
0.00001	168.669	117.801	5.133	5.133	5.133
0.0001	168.672	117.809	5.133	5.133	5.133
0.001	168.664	117.806	5.133	5.133	5.132
0.01	168.677	5.133	5.133	5.133	5.133
0.1	5.132	5.132	5.132	5.132	5.132

TABLE V: Data attack: turn star tracker off if quaternion difference is *bigger* than δ_{max} in columns or *smaller* than δ_{min} in rows. Consecutive failures=6. NO redundancy defenses. N/A if $\delta_{min} \geq \delta_{max}$

	δ_{max}			
δ_{min}	0.0001	0.001	0.01	0.1
0.00001	161.188	162.500	191.670	228.594
0.0001	N/A	162.813	192.109	228.641
0.001	N/A	N/A	194.488	226.909
0.01	N/A	N/A	N/A	229.570

TABLE VI: Data attack: turn star tracker off if quaternion difference is *bigger* than δ_{max} in columns or *smaller* than δ_{min} in rows. Consecutive failures=6. WITH redundancy defenses. N/A if $\delta_{min} \geq \delta_{max}$

	δ_{max}			
δ_{min}	0.0001	0.001	0.01	0.1
0.00001	156.508	151.699	74.316	80.627
0.0001	N/A	151.111	74.528	80.626
0.001	N/A	N/A	107.254	79.744
0.01	N/A	N/A	N/A	109.540

honest. Table I shows the attitude error after running the functional test without any redundancy and only one single honest star tracker. The table rows correspond to the limit checker settings of triggering an alarm if the data values changed by *less than* the threshold δ_{min}. The columns correspond with how many times in a row this threshold δ_{min} is seen before the limit checker marks the star tracker as malicious and turns it off. Most configurations in the table didn't impact nominal operations, but assuming δ_{min} of less than 0.1 or less than 0.01 led to mission failures when the consecutive failure count was too low. Thus, it would make sense for the mission operator to set the δ_{min} values to something like 0.001 or smaller. This is the most sensitive these thresholds could possibly be; in a real mission, the satellite developer would likely have to relax these thresholds even further to handle all nominal operations, making possible attacks that work within these scenarios easier to execute in a real mission setting. The results in Table II are similar given that the data is the same across star trackers, aside from noise applied to the quaternions that was sampled from a normal distribution with a standard deviation of 0.1.

On the other end, Table III and Table IV correspond to the limit checker settings of triggering an alarm if the data values changed by *greater than* the threshold δ_{max}, without and with redundancy respectively. The safest bet for detecting faults while still handling the nominal setting would be to set $\sigma_{max} >= 0.1$, and set the consecutive failure configuration to ≥ 6, though higher would likely be better to avoid being too sensitive for other nominal attitude maneuvers. As in the case with δ_{min} configurations, redundancy does not make a noticeable difference in the response to different thresholds since the data across star trackers is close to the same given the amount of data noise in the simulation.

2) Data Manipulation Attacks within Honest Thresholds: Next, we perform a manipulated data attack that stays within the allowable honest limit checker settings derived from the previous step. Given the results in Table I through Table IV, we show the limit checker consecutive failures setting of 6 for Table V and Table VI. However, we note that regardless of how many consecutive failures the operator allows, the attack is able to evade detection completely. To perform a stealthy attack, the malicious star tracker increments or decrements each quaternion value by the average threshold between the big and small threshold: $\delta_{attack} = \frac{(\delta_{min}+\delta_{max})}{2}$. It continues incrementing by δ_{attack} until it reaches the limit of quaternion value 1, then begins decrementing by δ_{attack} until it reaches -1. This attack leads to failure of every functional test, i.e., the satellite is never able to achieve its target orientation. Tables V,VI show the effectiveness of this attack. Despite tight constraints tuned for the honest setting, the stealthy attack achieves full mission compromise.

3) Limit Checker and Redundancy are Insufficient for Manipulated Data Attack: In the manipulated data attack, the limit checker flags a star tracker as anomalous only if its reported values are outside of normal (e.g., not between -1 and 1), or if the *change* in quaternion values are greater than a certain threshold between time t and $t+1$ or less than a certain threshold. Of course, this particular installation of the manipulated data attack could be defended against if we also implement a defense that checks that the pairwise difference between star trackers is within a certain range [23].

However, we argue that continuously adding more static limit checker policies in a "cat-and-mouse" game with the attack is generally ineffective. Whatever thresholds do get set in the limit checker, a malicious star tracker could just send data that is above the minimal threshold and below the maximal threshold. This motivates the need of a new anomaly detection technique for satellite operators that extends the capabilities of today's limit checker applications. For the scenario of pairwise differences, this would only work if only one of the three star trackers is compromised. Additionally, redundancy techniques often have a variety of sensor types, such as determining satellite orientation through fusing data from a star tracker, sun sensor, and earth tracker. And Park et al. note that the pairwise difference defense they propose may not hold up in stealthy attack scenarios [23].

Additionally, the limit checker has a maximum table size, as it must react to every single message and determine actions for every single watchpoint. The limit checker default size is not large enough to write multiple threshold values for every single sensor value and possible anomalous scenario. Even if the limit checker were large enough, writing possible thresholds for many different attack scenarios would put far too much burden on the developer for it to be a practical defense.

C. Time Manipulation Attack

Next, we investigate the Time Manipulation Attack where the compromised satellite peripheral withholds and then bursts (non-manipulated) data. In particular, we show that bursty communication from a malicious peripheral causes either 1) failure to successfully adjust the attitude of the satellite or 2) degradation of attitude maneuver performance on the satellite.

We also elaborate further in this section that the limit checker works poorly for time frequency thresholds.

1) Determining Timing Thresholds for Honest Case: Timing inconsistencies under no malicious adversary could be caused by different factors. One issue more likely in space than on Earth are radiation-induced faults [24]. Another issue is hardware aging [25]. A satellite developer may try detecting faults by monitoring the frequency at which data is sent. We thus test the limit checker at various levels of frequency at which to take action on the star tracker on differing levels of timing noise in the honest setting.

Tables VII and VIII show results of honest sensors at different noise levels and different limit checker thresholds. Timing noise in actual star tracker peripheral hardware is difficult to quantify, especially due to the variety of star tracker options at different price points, changes in timing behavior as hardware ages during a multi-year mission, and the difficulty of determining the exact impact of radiation on different hardware. Therefore, we test on a subset of possible honest noise values.

The columns of Tables VII and VIII correspond with how much timing noise the honest star tracker exhibits. The standard deviation is a half-normal distribution starting at 1 and truncated at 6. In other words, the burst amount can be between 1 (normal 1hz data timing) and 6, where the data would be sent in a burst of 6 data points in a row after 6 seconds.

Based on experimentation, 0.1 seemed like the most reasonable standard deviation to set an honest, non-faulting star tracker to. This noise amount was low enough that a burst of two data points or higher in the honest noise setting did not occur during testing, though it might experience multiple bursts over a multi-year satellite mission. The following timing attacks thus considered scenarios where the honest star trackers were set to a noise standard deviation level of 0.1. While timing jitter is less likely under low noise conditions, a developer would still want to tolerate small levels of noise so as not to trigger a star tracker shut-off/restart if the timing jitter is only temporary. Therefore, in the honest setting, a developer might not want to trigger an automated limit checker response after a burst of two data points at once, but may want to respond if there is a burst of three. We next perform manipulated time attacks and show that this burst of three limit threshold can lead to unacceptable mission slowdown, despite the attacker only being able to stealthily send bursts of two at a time.

2) Time Manipulation Attacks under Honest Thresholds: The manipulated data attack sends a burst of n data elements after n seconds, and no other data from the malicious star tracker in between those n seconds. Table IX and Table X show the effect of bursty attacks at different burst sizes when there is a single peripheral or multiple, respectively. The burst size in each column represents at what burst frequency a limit checker flags the star tracker. The rows correspond with the actual burst size of the attack occurring.

Even for attack burst size 2, performance degrades in both Table IX and Table X. Even when there is a majority of honest star trackers as in Table X, a burst size of higher than 3 allows the single compromised peripheral to degrade the satellite. The runtime for the functional test even with two other honest star trackers is increased by $\approx 3.4\%$. Due to realtime constraints of the satellite, even a 3.4% increase in runtime can sometimes be catastrophic for a mission.

3) Limit Checker and Redundancy are Insufficient for Manipulated Time Attack: Neither redundancy nor the limit checker are adequate to fully prevent the impacts of manipulated time attacks, while also handling temporary timing jitter in the honest setting, leading to a dilemma for the developer of whether to prioritize reliability or security.

Perhaps this could be fixed by having the limit checker check consecutive sets of bursts. For example, let us assume the operator thinks more than two messages per second should lead to immediate action. The operator could add custom code to check this, as we did in our limit checker tuning. They

TABLE VII: Honest case: turn star tracker off if there is a burst of n or more data points, where row is n. Column is standard deviation of burst size . NO redundancy.

Burst	Timing noise standard deviation		
threshold	0.1	0.25	0.5
2	5.127	4.828	172.859
3	5.127	4.828	5.189
4	5.127	4.828	5.189

TABLE VIII: Honest case: turn star tracker off if there is a burst of n or more data points, where row is n. Column is likelihood of a burst (standard deviation). WITH redundancy.

Burst	Timing noise standard deviation		
threshold	0.1	0.25	0.5
2	5.133	4.971	5.125
3	5.132	4.933	4.970
4	5.132	4.933	4.970

TABLE IX: Timing attack: turn star tracker off if there is a burst of n or more data points, where row is n. Column is what frequency triggers star tracker shutoff. NO redundancy. For example, the first row shows results for error area-under-curve of a bursty attack of size 2, where the burst frequency threshold shuts the star tracker off if it bursts in size 2 (leftmost column) to size 4 (rightmost column).

Burst	Burst limit checker threshold		
size	2	3	4
2	191.979	5.645	5.645
3	175.372	175.376	38.605
4	179.370	179.383	179.382

TABLE X: Timing attack: turn star tracker off if there is a burst of n or more data points, where row is n. Column is what frequency triggers star tracker shutoff. WITH redundancy.

Burst	Burst limit checker threshold		
size	2	3	4
2	5.176	5.170	5.170
3	5.193	5.192	5.199
4	5.232	5.235	5.236

may deem one or two bursts in a row as honest and no action is needed, but they may want to take action if there are more bursts in a row. Unfortunately, consecutive failures for timing attacks will not be detectable due to the way the cFS limit checker is architected. Given that the limit checker must evaluate every star tracker message if it is subscribed, it must return a pass or fail for each data point when a watchpoint is encountered. This means that if $n-1$ data points are seen in a second, the limit checker counts this as a pass. When n data points in a burst are seen, this is one failure. But then when the next data point is seen, this is n seconds after the previous burst, which means that the limit checker would return a pass, and therefore the consecutive failure count returns back to 0. Therefore, the satellite developer can only trigger action upon a single burst, or trigger no action at all.

D. More Limit Checker Security Challenges

The NASA cFS limit checker is set to listen to the message id's that a developer specifies. It watches for these message id's, and compares these to threshold values. As the table of watchpoints scales up, simply having watchpoints and possible responses may lead to performance impacts, especially for smaller satellites. In the case of our limit checker thresholds, we had to write custom comparison functions for both checking the difference between data values and the timing frequency of values. By default, the limit checker is only designed to handle very simple checks, such as checking that the star tracker quaternion values are in range. Writing these custom functions, not to mention thinking through all possible peripheral attacks, becomes burdensome for the satellite developer.

The limit checker becomes even more "limited" when this test setting is scaled up into reality. With the redundancy and the custom thresholds we included in our limit checker watchpoint table, we already used up 51 out of 176 possible watchpoints for the default cFS watchpoint table size, or almost a third of watchpoint slots. This does not yet include watchpoints for any other subsystem, such as the power system or thermals, both of which require careful monitoring so as to not lead to physical destruction of the satellite. Thus, not only is using the limit checker to handle possible security issues burdensome on the developer, there may be size, power, and runtime constraints on constructing a more carefully-tuned limit checker.

VII. Future Work

Future work is needed to build satellite architectures that are more resilient to supply chain attacks.

a) Limitations of existing mitigations in other control systems and robotic vehicles: One possible approach to mitigating the malicious peripheral risk is vendor redundancy, or using sensors (e.g., star trackers) from multiple vendors. Vendor redundancy is an important step in mitigating this problem, though it is not sufficient. Vendor redundancy reduces the likelihood that a majority or all sensors are compromised. However, if there is one singular sensor from the malicious vendor, the attacks in this paper are feasible without new mitigations.

And while thorough supply chain verification by all satellite owners is ideal, the current state of the space economy makes this practice unlikely to exist uniformly. Previous work supports that the space industry may be underprepared and lacking incentives for such a task [17], [18].

b) Future work directions: Future work should consider the inherent issues in threshold-based anomaly detection such as that of the limit checker. Work can use inspiration from research on attack detection and recovery from other robotic vehicles, while adjusting to this specific highly-privileged supply chain attack threat model. Satellite-specific work to resolve this malicious peripheral dilemma can possibly use the specific features of this domain to their advantage, such as the redundancy of sensors in these control systems, as well as the long mission timespan to perform long-running tests. Future work could also test on real hardware to measure the power and runtime overhead of whatever new solutions are posed.

VIII. Conclusion

In this work, we demonstrated that common existing spaceflight software platforms are ill-equipped to handle malicious satellite peripherals. In particular, we demonstrated a dilemma for space systems in which either the system deploys countermeasures against malicious components and suffers potentially unacceptable nominal performance, or the system cannot survive malicious components. We show this along two axes: manipulation of data and manipulation of time, and show that these axes of attacks avoid detection while leading to unacceptable slowdown or mission failure. We demonstrated this satellite peripheral dilemma on the core Flight System from NASA in an state-of-the-art emulated, closed-loop environment.

References

[1] USGS, "Number of Commercial, Government-Civil Satellites Launched," 2025, https://www.usgs.gov/media/images/number-commercial-government-civil-satellites-launched.

[2] Iridium, "Iridium satellite time & location," https://www.iridium.com/satellite-time-location/.

[3] Maxar, "Next-Generation Weather Satellite Technology," https://www.maxar.com/splash/weather.

[4] P. Labs, "Planet Labs Homepage," https://www.planet.com/.

[5] O. W. in Data, "Cost of Space Launches to Low Earth Orbit," https://ourworldindata.org/grapher/cost-space-launches-low-earth-orbit.

[6] G. Brunetti, G. Campiti, M. Tagliente, and C. Ciminelli, "Cots devices for space missions in leo," *IEEE Access*, 2024.

[7] P. Tedeschi, S. Sciancalepore, and R. Di Pietro, "Satellite-Based Communications Security: A Survey of Threats, Solutions, and Research Challenges," *Computer Networks*, vol. 216, 2022.

[8] S. Jero, J. Furgala, M. A. Heller, B. Nahill, S. Mergendahl, and R. Skowyra, "Securing the Satellite Software Stack," in *Workshop on Security of Space and Satellite Systems (SpaceSec)*, 2024.

[9] B. Cyr, Y. Long, T. Sugawara, and K. Fu, "Position Paper: Space System Threat Models Must Account for Satellite Sensor Spoofing." in *Workshop on Security of Space and Satellite Systems (SpaceSec)*, 2023.

[10] W. Xu, X. Chen, M. Anderson, S. Drager, and F. Kong, "Recovery-guaranteed sensor attack detection for cyber-physical systems," *2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 324–336, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:279172199

[11] P. Lu, L. Zhang, M. Liu, K. Sridhar, O. Sokolsky, F. Kong, and I. Lee, "Recovery from adversarial attacks in cyber-physical systems: Shallow, deep, and exploratory works," *ACM Computing Surveys*, vol. 56, no. 8, pp. 1–31, 2024.

[12] A. Li, J. Wang, and N. Zhang, "Software availability protection in cyber-physical systems," in *34nd USENIX Security Symposium (USENIX Security 25)*, 2025.

[13] P. Dash, E. Chan, and K. Pattabiraman, "Specguard: Specification aware recovery for robotic autonomous vehicles from physical attacks," *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271962902

[14] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "Pid-piper: Recovering robotic vehicles from physical attacks," in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2021. [Online]. Available: https://doi.org/10.1109/DSN48987.2021.00020

[15] G. Engineering and N. Technology Directory, "core Flight System," 2025, https://etd.gsfc.nasa.gov/capabilities/core-flight-system/.

[16] ——, "core Flight System Limit Checker Application Version 2," 2025, https://software.nasa.gov/software/GSC-16010-1.

[17] R. McAmis, G. Haas, M. Sim, D. Kohlbrenner, and T. Kohno, "Short: Unencrypted Flying Objects: Security Lessons from University Small Satellite Developers and Their Code," in *Symposium on Vehicle Security and Privacy (VehicleSec'25)*, 2025.

[18] J. Willbold, M. Schloegel, M. Vögele, M. Gerhardt, T. Holz, and A. Abbasi, "Space Odyssey: An Experimental Software Security Analysis of Satellites," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023.

[19] J. Pavur and I. Martinovic, "Building a launchpad for satellite cyber-security research: lessons from 60 years of spaceflight," *Journal of Cybersecurity*, vol. 8, no. 1, p. tyac008, 2022.

[20] G. Falco and N. Boschetti, "A security risk taxonomy for commercial space missions," in *ASCEND 2021*, 2021, p. 4241.

[21] P. W. Kenneally, S. Piggott, and H. Schaub, "Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework," *Journal of Aerospace Information Systems*, vol. 17, no. 9, 2020.

[22] J. R. Wertz, Ed., *Spacecraft Attitude Determination and Control*, ser. Astrophysics and Space Science Library. Dordrecht: Springer Netherlands, 1978, vol. 73. [Online]. Available: https://link.springer.com/10.1007/978-94-009-9907-7

[23] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2735960.2735984

[24] R. Ecoffet, "Overview of in-orbit radiation induced spacecraft anomalies," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1791–1815, 2013.

[25] M. T. H. Anik, S. Guilley, J.-L. Danger, and N. Karimi, "On the effect of aging on digital sensors," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*. IEEE, 2020, pp. 189–194.

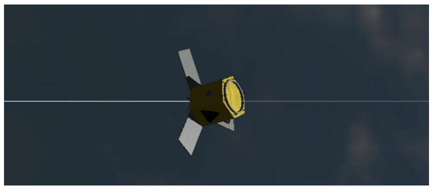
Appendix

A. Example Attack Visualizations

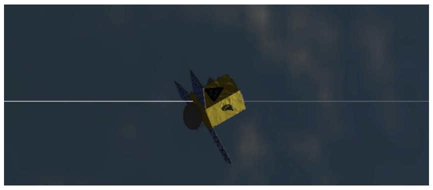
In order to conceptualize the attacks, in this section, we provide a set of visualizations of the impact of the attacks. Figure 2 shows a visualization of the simulation under a manipulated data attack, as well as the honest scenario, without any defenses. The attack clearly causes the satellite to tumble uncontrollably, whereas with an honest star tracker, the satellite can always orient itself properly relative to Earth.

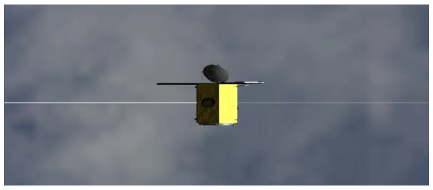
Additionally, in Figure 3, we show the difference between the current and target quaternion values over time. In particular, Figure 3g and Figure 3h show an example of the manipulated data attack when the satellite uses multiple star trackers or only one star tracker, respectively. However, in either case, the quaternions never converge to the goal orientation.

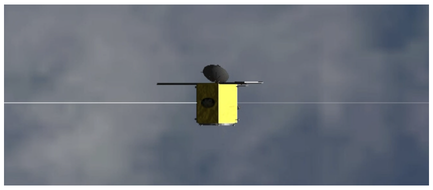
In the case of a burst size of 3 with no defenses in place, interesting behavior is visible through time series plots. Figure 3b and Figure 3e show the difference across time between the current quaternion (orientation) values for the satellite and the target quaternion values, which can be seen as an error metric simpler than the one we describe above. With redundancy, there is slight performance degradation, as convergence happens longer compared to nominal timing conditions. When there is no redundancy, the satellite achieves "marginal stability", meaning it is not completely unstable or completely stable. This could still cause negative effects, such as requiring the reaction wheels to do extra physical work to attempt to achieve convergence, or failure of the mission entirely due to the inability to orient properly.


(a) Malicious star tracker: Right after attack begins.

(b) Malicious star tracker: 3s after attack begins.


(c) Malicious star tracker: 6s after attack begins.


(d) Honest star tracker: 0s into mission.


(e) Honest star tracker: 3s into mission.

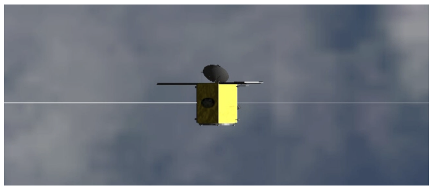
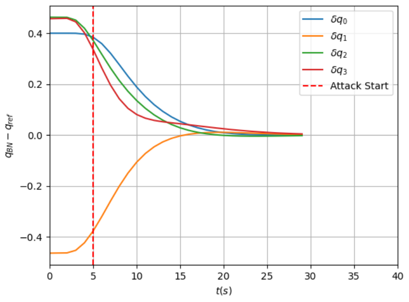
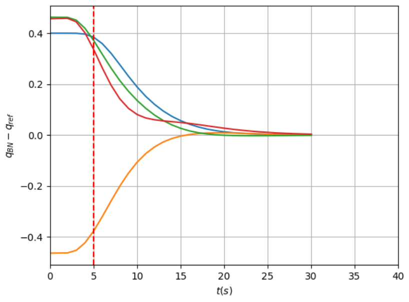

(f) Honest star tracker: 6s into mission.

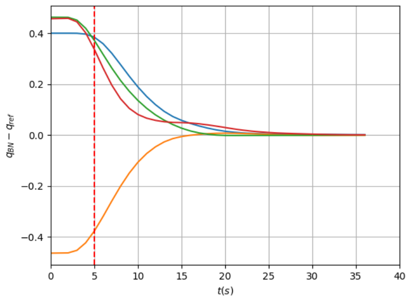
Fig. 2: A malicious star tracker can cause a satellite to tumble using the Data Manipulation attack (top), whereas, with an honest star tracker, the satellite would maintain its attitude (bottom).



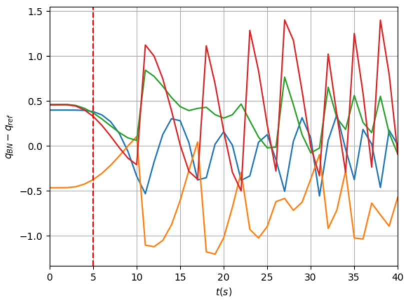
(a) Honest. δ_{max}=0.1. No redundancy



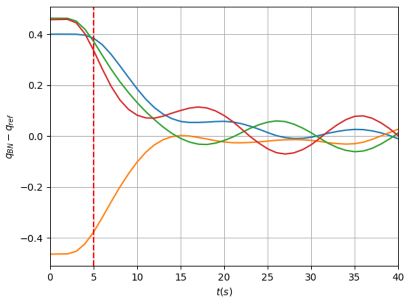
(b) Burst size 3. With redundancy. No LC thresholds.



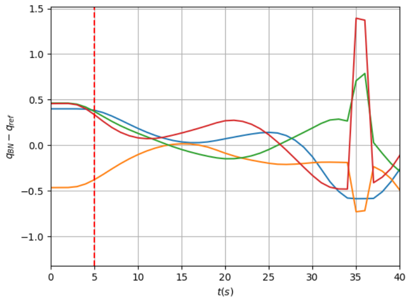
(c) Burst size 6. With redundancy. No LC thresholds



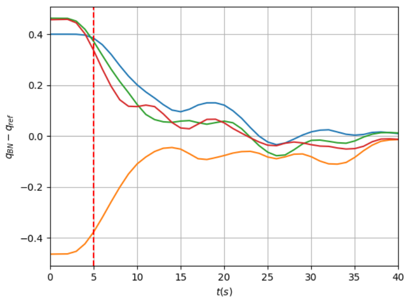
(d) Honest. δ_{max}=0.0001. No redundancy



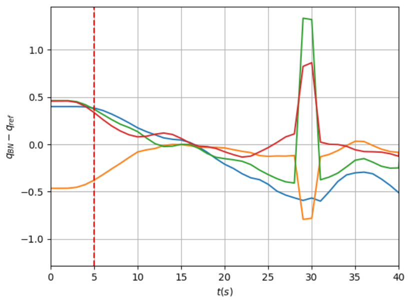
(e) Burst size 3. No redundancy. No LC thresholds.



(f) Burst size 6. No redundancy. No LC thresholds.



(g) Data Manipulation. δ_{max}=0.1. Redundancy



(h) Data Manipulation. δ_{max}=0.1. No redundancy.

Fig. 3: Time versus attitude error of different cases of attacks. The four lines are the four quaternion values making up the satellite's orientation. The attitude error for each quaternion is the current quaternion value subtracted by the target quaternion value. The red dotted vertical line marks when the attack starts. If the quaternion lines end before the x-axis limit, this means that it passed at that point and exited the test.