Securing the Satellite Software Stack

Samuel Jero, Juliana Furgala, Max A Heller, Benjamin Nahill, Samuel Mergendahl, Richard Skowyra
MIT Lincoln Laboratory
{samuel.jero, juliana.furgala, max.heller, bnahill, samuel.mergendahl, richard.skowyra}@ll.mit.edu

***Abstract*—Satellites and the services enabled by them, like GPS, real-time world-wide imaging, weather tracking, and world-wide communication, play an increasingly important role in modern life. To support these services satellite software is becoming increasingly complex and connected. As a result, concerns about its security are becoming prevalent.**

While the focus of security for satellites has historically been on encrypting the communications link, we argue that a fuller consideration of the security of satellites is necessary and presents unique challenges. Satellites are becoming increasingly accessible to attackers–thanks to supply chain attacks and Internet connected ground stations–and present a unique set of challenges for security practitioners. These challenges include the lack of any real ability for a human to be physically present to repair or recover these systems, a focus on safety and availability over confidentiality and integrity, and the need to deal with radiation-induced faults. This work characterizes the cyber threats to satellite systems, surveys the unique challenges for satellite software, and presents a future vision for research in this area.

I. Introduction

Satellite constellations offer a suite of services crucial to the modern world. Position, navigation, and timing (PNT) constellations, such as GPS, provide invaluable support to a diverse collection of systems ranging from consumer smartphones and ATMs to commercial aircraft and military weapons systems. Earth imaging has revolutionized weather forecasting and planetary science. More recently, commercial imagery of military conflicts in near real time has enabled unprecedented visibility into active warzones [54]. The RF-sensing constellation operated by HawkEye 360 has even detected and publicized the location of Russian military jammers operating in Ukraine during the ongoing conflict there [59]. Commercial satellite communications have opened Internet and voice/data capabilities to previously isolated remote regions, and can continue to operate when terrestrial infrastructure is unavailable, damaged, or destroyed.

This importance is not lost on adversarial actors. GPS jamming has been observed in 2023 across several conflict areas in the Middle East, and has interfered with the inertial navigation systems of commercial flights in the region [48]. Russia launched a cyberattack against Viasat's KA-SAT satellite communications service immediately prior to its invasion of Ukraine, with wide-reaching loss of communications capability [28]. There have been ongoing attempts, also allegedly by Russia, to jam Starlink services over Ukraine as well [25]. More broadly, the use of cyber weapons against space systems is increasingly recognized as a credible counterspace threat [35] that can cause irreversible harm to satellites. This is especially concerning given their cost, long operational lifetimes, and the difficulty in replacing them should they be lost.

Historically, satellite security efforts have focused on encryption of the communication link. This is necessary, but clearly no longer sufficient. The complexity of satellite software continues to grow as industry moves toward ever more 'software-defined' systems that draw on a long tail of dependencies. These are vulnerable to software supply chain attacks whose effects may not manifest until long after launch and any hope of remediation. Ground stations themselves are regularly Internet-connected, opening avenues for malware to make its way into the trusted perimeter and potentially issue seemingly authorized commands over the encrypted link.

Securing space systems comes with unique challenges not seen in other embedded domains. Of course, in some ways satellites are indeed traditional embedded systems. They contain a small computer connected to custom hardware with a software stack dedicated to performing a specific task, similar to a drone, car, or smart camera. For these kinds of common embedded and Internet-of-Things (IoT) systems, the software security challenges have already been well-studied in the literature [3], [13], [22], [32], [53], [56]. However, satellites are distinct in their sheer remoteness and need to operate in harsh orbital environments. This presents several challenges to securing them against cyber threats.

First is the lack of human accessibility. Unlike terrestrial embedded and IoT systems, where a human is able to repair or reboot the device as a last resort, satellites are launched into space with no ability for humans to physically access, reboot, repair, or replace components. As a result, safe modes and autonomous recovery approaches are crucial to eventually reestablishing remote operator control. This is particularly critical in the context of cyber attacks. Operators must reassert control over compromised satellite software and eject the attacker, without having physical (or even continuous remote) access.

Second is the paramount need for system safety and availability. Unlike most terrestrial embedded and IoT systems,

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported under Air Force Contract No. FA8702-15-D-0001. 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 U.S. Air Force.

Workshop on Security of Space and Satellite Systems (SpaceSec) 2024
1 March 2024, San Diego, CA, USA
ISBN 979-8-9894372-1-4
https://dx.doi.org/10.14722/spacesec.2024.23057
www.ndss-symposium.org

satellites turn the traditional Confidentiality, Integrity, and Availability (CIA) triad on its head; Safety (not physically damaging the spacecraft) and Availability (continuing to perform the mission and be available) are often prioritized by satellites over Integrity and Confidentiality. In this regard, they are similar to safety-critical cyber-physical systems [4], [57]. Importantly, this emphasis on availability extends to any security systems for a satellite: a solution that crashes the satellite or aborts processing is simply unacceptable.

Third, satellites operate in a harsh radiation environment that can induce faults in otherwise correct hardware. Outside the protection of earth's atmosphere, cosmic radiation regularly strikes processing circuitry on satellites. Radiation-induced faults can manifest as transient bitflips or more long-lasting errors. While hardware design can mitigate some faults and prevent permanent failure due to others, some faults will still regularly impact the software in the system and necessitate software-based mitigation or recovery strategies [19].

Finally, as identified by prior work [9], [60], most satellites operating today rely on legacy flight software attuned to operational needs and not designed to operate through cyber attack. In some cases, the software has been designed for reusability and modularity, (e.g., NASA's core Flight System (cFS) [8]), but in other cases, it is akin to a single large state machine that senses and actuates satellite subsystems. Basic security principles like the *principle of least privilege* [40] do not figure into the design of these systems.

Due to these challenges, we argue that securing satellite software deserves special attention by the security community and cannot easily rely on prior work in the terrestrial embedded domain. This is true both for securing legacy systems, and for building new flight software designed to operate in a contested cyber environment.

After providing some brief background (Section II), the remainder of this paper makes the following contributions:

- We characterize the cyber threats to satellite systems, emphasizing the importance of assuming compromise and exploring supply chain attacks as a particularly likely threat vector. (Section III)
- We survey in detail the challenges to current space systems including the lack of human accessibility, the importance of safety and availability, the impact of radiation, and the difficult-to-secure design of existing software. (Section IV)
- We propose a vision for the future and suggest a research roadmap to achieve that vision. (Section V)

II. Background

In this section, we provide background on satellite flight software, radiation-induced faults, and survey existing satellite security work.

A. Satellites and Flight Software

Satellite flight software is tasked, at minimum, with processing commands from the ground station and operating the hardware peripherals on the space vehicle (called the bus) in order to maintain safety-of-flight. The bus provides mission-agnostic services and infrastructure, such as managing the power and thermal environments, maintaining a command and data link to the ground, attitude control (pointing direction), and orbital maneuvers (if the satellite is equipped with propulsion). Flight software manages and coordinates these cyber-physical functions. It serves a similar role to avionics software in aircraft and is held to similar standards of assurance and correctness.

One key difference between satellite flight software and avionics, however, is that satellite operators may be unavailable for large intervals of time due to line-of-sight communication requirements. Depending on the orbit, it is not uncommon for a satellite to transit over a ground station for only ten to fifteen minutes out of every ninety. Flight software must therefore autonomously manage crucial resources like power and heat, perform complex orbital maneuvers, maintain orientation, and execute stored tasking commands.

Flight software may also, to varying degrees depending on the specific satellite, manage the payload(s) that provide some service to satellite operators (e.g., communications, sensing, imaging, PNT). Some satellites tightly integrate payload and bus, for example to enable bus-steered optical sensors. In these architectures, flight software is often also responsible for payload management [14]. Other approaches treat the bus as a separate, shared infrastructure provider for a series of independent hosted payloads, many of which may be from different organizations and perform differing unrelated tasks (e.g., shared ESPA-style buses [12]). Bus flight software in these systems is limited to providing safety-of-flight and common services such as power and propulsion.

Initially, spaceflight software and hardware were unique, custom components designed solely and specifically for a single mission. Today, we see spacecraft leverage specialized but common hardware platforms with well-known architectures, such as the RAD750 (Sparc) and the Xilinx Zynq7000 (armv7). Sometimes these are Commercial Off-the-Shelf (COTS) components; other times, they are modified specifically for space missions. Modern spaceflight software also tends to build upon standard, full-fledged operating systems such as Linux, FreeRTOS, VxWorks, and Green Hills' Integrity rather than single-use, bare-metal software. SpaceX, for example, famously uses Linux on their satellites [55].

Additionally, spaceflight control software itself is becoming increasingly reusable. Realizing that many of their missions required overlapping functionality, NASA developed a modular, layered spaceflight software stack called the core Flight System (cFS) [8]. The foundation of cFS is the core Flight Executive (cFE), a portable, platform-independent framework that provides an application runtime environment with services common to most flight applications. Building on the cFE, the application layer of cFS hosts reusable software components that different missions can deploy for common mission functionality. Other satellite flight software platforms exist and are similarly modular. While modular software is an improvement over the prior state, none of this software is yet designed with isolation between modules or consideration of basic security principles. Sadly, not even all new missions today use modular software, with many spaceflight missions continuing to choose monolithic, single-use software deployments since the one-time cost is often perceived to be lower.

B. Cosmic Rays and Fault Tolerance

A challenge that is unique to space systems is the need to operate in the presence of cosmic rays that can alter hardware states [6]. Cosmic rays include a multitude of high-energy particles from a variety of sources, which may impart charge on circuit elements, disrupting stored and transient values. This induces bitflips in SRAM or flip-flops that cause calculations to fail or return erroneous results. The total set of possible effects are complex and can include much more destructive behavior, but we will focus on Single Event Upsets (SEUs) — so-called *soft errors*— for this discussion, as system designers should avoid parts likely to experience more destructive effects. The actual soft error rate depends on many factors: the environment, shielding around a device, the chip fabrication process, and low-level features of the circuit.

The outcome of an unmitigated SEU can vary. In many cases, the upset is simply masked by a future overwrite with no meaningful lasting impact. However, an SEU can also lead to incorrect computations or memory state corruption, which can be extremely difficult to detect in time to mitigate. This so-called silent data corruption can threaten system safety and availability. Detecting, mitigating, and designing against it is critical for any software that will operate outside the atmosphere.

Soft errors must be accounted for in spaceflight software because fabrication processes are no longer able to guarantee robustness to particle strikes (generally due to smaller circuit features requiring less charge to affect operation). Radiation hardened parts now implement redundancy at other levels of the design, either at the cell library level [17], the CPU [24], or somewhere in between. Unfortunately, the cost of this redundancy is prohibitive for commercial applications. These remain niche parts that fit only the highest of budgets. Instead, it is common to use COTS processors for limited-lifespan missions, accepting some rate of upsets and eventual failure due to long-term effects.

C. Existing Satellite Security Work

Historically, work securing satellite systems has focused on the communications link between the satellite and the ground station. The security of this link [37], [47] and its physical layer medium [31], [42] have received significant attention. Other work has reported on particular satellites trusting in obscurity as their only security protection [36].

An area of more recent focus has been the security of analog sensors on satellites [61]. In particular, there has been significant effort to prevent satellite communications (SATCOM) interference [31]. More recent work has studied the ability for an attacker to *spoof* analog signals, rather than only impact their availability [10].

Because satellite software is mostly implemented in C/C++, where memory safety vulnerabilities are a particular concern, other academic work calls for standard software security improvements for satellite software [9], [60]. These works also emphasize the importance of designing satellite software with security and reliability in mind [9], [41].

There are also frameworks for describing security gaps in a software stack, such as MITRE's ATTACK project [51], MITRE's CAPEC framework [52], and NIST's OSCAL [33]. A recent entry in this category that focuses on space systems is Aerospace's SPace Attack Research & Tactic Analysis (SPARTA) guidelines [1].

III. Cyber Threats to Satellites

Satellites are part of a larger space system architecture that collectively provides a service or conducts a mission. Other components of this system are the ground control station, potentially user terminals (e.g., for satellite communications), and potentially other satellites in a constellation. For the purpose of this work we focus on the threats to the satellite itself, not on malicious ground stations or constellation members.

A number of existing works provide extensive attacker models for satellites [60] or focus on particular aspects of the threat to satellites [10]. Here, we focus on motivating what we argue should be a core design assumption when addressing the unique challenges to satellite cybersecurity: that *a cyber-attack has already succeeded*. That is, satellite software should be able to operate through, and recover from, a successful compromise of one or more software components. It is not enough to rely on hardening external interfaces to the system.

A software supply chain attack is a compelling example of why that assumption should hold. The software dependency base of even very simple flight software is large and often includes familiar packages such as the Linux operating system or popular high-performance computing libraries (e.g., for image or signal processing). There will also be a collection of common drivers not unique to the satellite itself, such as for serial and other data ports. This is potentially fertile ground for attackers to compromise the software supply chain; a technique that has been used to great effect in recent years on terrestrial enterprise networks (e.g., SolarWinds [2], NotPetya [15]).

Consider a software supply chain attack leveraging malicious drivers that are provided as part of a 3rd party vendor's integration package. Malicious signed drivers have been used in recent terrestrial cyber attacks [30]. Such drivers could provide a highly privileged foothold for autonomous malware that bypasses the need to break link encryption entirety.

The intent of this paper is not to argue that this particular example is more or less likely than other cyber threats to space systems. However, it is in many ways a worst-case scenario from the perspective of defenders and is rooted firmly in the realm of the possible. Thus, we argue that spaceflight software should be designed from the start to operate under the assumption that parts of the satellite's software stack may be compromised from the beginning (although they may not become actively malicious immediately).

IV. Current State of the Art

In this section, we examine the current state of the art around our major challenges to satellite software: lack of human accessibility, an emphasis on safety and availability, radiation, and insecure legacy software.

A. Lack of Accessibility

Since satellites are effectively inaccessible to humans once launched, satellites are normally configured to recover autonomously back to a known-good state in the event of a critical failure. A common approach is building one or more "safe modes" into the system that provide only extremely simple safety-of-flight functionality: point the solar panels at the sun to keep the satellite power positive and point the antenna at the ground to enable communication. By limiting functionality to only the most critical systems, the majority of potentially faulting components can be kept offline.

These "safe modes" are often supplemented with a boot-time mechanism that first attempts to boot a preferred system image and, if that fails, tries a series of less-preferred images [14], [45]. This mechanism also enables safe update of the satellite's software by replacing one system image and attempting to boot it while preserving a known-good image, similar to Android's A/B boot [49].

Unfortunately, offering a "safe mode" as satellites do today is insufficient when considering malicious attackers. Adversaries may be able to disable the "safe mode", prevent it from being actuated, or persist in the reduced software stack that still remains online.

To truly remediate compromised spaceflight software, an independent, isolated high-assurance mechanism not accessible in any way to the primary software stack is required. We refer to this as a root-of-recovery (RoR). At minimum, the RoR must be capable of preserving communication with the ground, receiving software updates, and deploying them to re-assert control over a compromised satellite. It may also be desirable to move other safe-mode functionality into the RoR, but this must be balanced against adding complexity that can make high assurance operation more difficult to prove.

A root-of-recovery must be a highly-secure component that is strongly isolated from the rest of the system; to operate correctly, it needs to be unaffected by system compromise. It is itself also a valuable target for attackers and must remain inviolate. A compromised RoR is effectively an attacker-controlled bootloader. A separate processor, an FPGA IP core, or a secure processing enclave are all potential approaches. Fortunately, there is substantial prior work that can be drawn from when considering how this type of isolation can be provided [21], [29], [58].

In addition to whole-system software updates, it may be valuable for a RoR to include more advanced features that minimize disruption to the overall mission of the satellite. These could include software system reconfiguration to mitigate attacker-controlled components, or finer-grained reset, update, and recovery, for example.

B. Emphasis on Safety and Availability

Satellite software has a unique emphasis on Safety and Availability over other security goals, based partially on the lack of physical availability. In particular, the most important security property for a satellite is safety. This is the idea that the system should not harm itself or other satellites. This could happen, for example, by accidentally pointing a camera at the sun or by colliding with another satellite. We see similar concerns in terrestrial cyber-physical systems, such as the power grid.

After Safety, Availability is the next most important property for a satellite. This is a critical concern for satellites because no human will have access to the satellite once launched so there is effectively no ability to remotely recover a satellite that has lost power or communications. The only way to repair the satellite is via its computer system. Additionally, the missions being performed by satellites—PNT, imaging, communications—are extremely vital and minimal downtime is required.

With Safety and Availability guaranteed, satellites would also benefit from Integrity, especially the knowledge that the satellite is running the correct and unmodified software image. Confidentiality, if considered at all, is a nice to have, at least within the satellite itself.

In contrast, cybersecurity has historically focused on violations of Confidentiality, Integrity, and Availability (CIA) [40], in that order. Of most importance is Confidentiality followed by Integrity. Availability is a distant third, often only provided if the system is not under attack. Notice that this is almost exactly opposite how satellite systems rank the importance of their goals.

This mismatch has significant impact. For example, computer security techniques generally default to crashing the process or the system if an attack is detected. For satellites, this approach is completely unacceptable.

C. Radiation Faults

As discussed in Section II, the space environment is generally hostile to functioning microelectronics, inducing periodic faults which would ideally be detected and handled. Achieving this to any extent requires a whole-system design approach, starting with processor selection.

Many modern reliability-oriented application processors (e.g., NXP Layerscape® 1046A [34]) are able to provide now-standard protections which help limit the potential impact of radiation-induced faults:

- Single-error correct, double-error detect (SECDED) codes for data caches and main memory [16]
- Simple parity checking for read-only caches such as instruction caches and translation lookaside buffers
- Hardware memory patrol scrubbers able to read and correct data errors in memory
- Memory protections at the CPU—e.g., a memory management unit (MMU) or memory protection unit (MPU)
- Memory protections within bus infrastructure—e.g., a system MMU (SMMU) or Input/Output MMU (IOMMU)
- Hardware watchdog timers (WDTs) to help detect loss of liveness

However, while these features are available, they are not all broadly adopted. This is often the case because designers choose parts with fewer features, but more extensive flight heritage. For example, the Xilinx Zynq7000 series SoC FPGA is still often selected for use despite having a $\sim$18x worse SEU rate than its more modern FinFET-based Zynq UltraScale+ counterpart [18], no error correcting memory, no SMMU, and no memory scrubber.

This choice seems illogical, but stems from often sensible reasons. First, The Zynq UltraScale+ is not simply the newer generation—it's a vastly different device and its many benefits come at both monetary and power costs. Second, more known errors can be attractive over fewer unknown errors, especially depending on the impact of those errors. This is particularly true given that the amount of damage that can occur without detection is quite high, especially with unprotected memory and/or cache.

Ultimately, designers select thresholds for errors that can be detected and recovered, e.g., one per week or month, and use this to select a processor. This ultimately means that satellite software must handle soft—i.e., recoverable—errors, with some nonzero frequency.

These errors have been studied extensively for decades [6]. The breadth of functional effects is extensive and there is no surefire detection method. A system would ideally be expected to fail safely and not to a vulnerable or significantly degraded state, preferably resorting to some restorative action such as rebooting. However, even that is extremely challenging and many have attempted to improve the failure detection rate of software [20] [43] with some success. These techniques are challenging to apply for even moderately complex software, but may be able to be applied selectively to especially sensitive code regions, for example, operations which manipulate page tables.

How these errors impact more complex software systems and how they might impact security guarantees onboard spacecraft is an open research question.

D. Insecure Software

Much existing satellite software is poorly suited for operating in a malicious environment. For example, NASA's cFS [8], although designed to be modular from a software development perspective, runs modules as threads in a shared address space, allowing malicious attackers to trivially move between them. Similar issues are prevalent in common monolithic operating systems—once one part of the operating system is compromised, the entire operating system and any applications running on it are trivially compromised.

One of the largest issues in satellite software is the pervasive use of memory-unsafe languages like C/C++ in satellite software. Memory safety vulnerabilities are one of the largest classes of issues in modern software [7], [50] and although a host of mitigations have been developed [46], these are rarely used in space systems.

Another issue is that this software is usually designed without regard to the *principle of least privilege* [40]. Software running as root is pervasive [5], [9]. Even when software is modular, it may not provide isolation between modules, as in cFS. Further, in commonly used operating systems like VxWorks and FreeRTOS, isolating userspace from the kernel is optional and not always enabled [38].

Prior work has covered many of these issues and suggested at least partial solutions [9], [60].

V. A Secure and Resilient Software Stack

In this section, we provide a vision for a future secure satellite software stack and outline key research required to achieve this vision.

We argue that the satellite software stack of the future needs to carefully address the challenges we have laid out above. To cope with a lack of human accessibility, it needs not only safe modes but a root-of-recovery that can enable operators to reassert control over a compromised satellite. In order to enable this, the operating system and flight software are going to need to become security-focused and modular, with strong isolation between components. To support safety and availability as paramount concerns, broader solutions for dealing with faults beyond simply crashing or rebooting will be needed, especially where malicious behavior is involved. Future software stacks also need to handle radiation faults smoothly, with some confidence in how these faults can impact the system and its security. Finally, while much of the future software stack needs to be carefully designed for security—following the principle of least privilege and ideally be written in memory safe languages like Rust—it will also inevitably need to interoperate with existing legacy components.

Satellite software today is a long way from this vision in many areas. Research is needed on a wide array of topics before this vision can become reality. In the remainder of this section, we explore a few of the key areas where research is most urgently needed.

A. Isolated, Modular Systems

At the core of this vision are software systems that consist of isolated modules. This enables everything from operating systems that handle compromise well, to flight software that can restart failed components, as well as many root-of-recovery designs.

Although the principle of least privilege is nearly 50 years old [40], developers still struggle to implement it effectively, especially for embedded systems. Given the importance of safety and availability to satellite systems and the need to provide an ability to reassert control over a compromised satellite, we argue that operating systems used in these environments need to isolate their functionality into multiple components. The microkernel design, in which a very small kernel provides only the functionality that requires kernel mode and userspace services provide the rest of the operating system functionality, seems like a good template.

We have implemented a microkernel-based operating system around the seL4 [23] microkernel. It has about 17 services and provides enough functionality to run the core components of NASA's cFS [8] software. Despite traditional concerns about the performance of microkernel systems, we have found its performance to be acceptable for cFS and other flight software. Prior work has also demonstrated microkernel based systems for embedded systems and was able to achieve similar performance to real-time Linux on microbenchmarks [21].

Further research is needed on secure operating systems for satellites. In particular, research is needed to understand how best to design these operating systems for satellites and how to separate functionality into ideal components that minimize the impact of compromise while still providing good performance.

In addition to the operating system layer, isolation is also desirable within the flight software itself. In fact, much flight software is already separated into modules and uses a data bus to communicate between these modules [8]. What is missing today is a focus on isolation between these modules. Unfortunately, recent work emphasizes that it is not merely enough to draw some isolation boundaries between modules and claim security: the data crossing those boundaries and assumptions being made across those boundaries matter [26].

We are also exploring a modular satellite flight software system design that provides isolation between components. This system leverages a software data bus to connect multiple modules that are implemented as separate processes. We have found this approach to be performant enough for some basic example satellite missions, at least in a software testbed.

Further research is needed here as well, especially around alternative designs for flight software, isolation boundaries and methods, and techniques for dealing with the compromise of components that many other components depend on.

A final challenge related to isolation is the interaction between isolation and particular classes of debugging tools that frequently show up on satellites. In particular, many satellites systems have the ability to read and write arbitrary system memory at the command of the ground [8], functionality which appears to be used for debugging and repair of satellites. We speculate that such functionality is part of what enables NASA to recover its spacecraft from significant failures [11]. Such functionality is completely incompatible with isolation, forming a trivial isolation bypass. Research is needed to understand how to provide sufficient debugging and repair functionality while also providing security and isolation.

B. Assurance Technologies

Given the significant expense and lead time associated with launching a satellite and the critical missions performed, we argue that satellite software is a promising area to explore future technologies for providing assurance about software systems.

One particularly promising technology is formal verification, which attempts to mathematically prove properties of a software system. As perhaps the strongest means of assurance we have today, we think there is significant value in applying it to satellite systems. This is especially true for critical aspects of the system like isolation between components and between the main system and the root-of-recovery.

While verification of control algorithms is occasionally done for satellites, we are unaware of any attempts to verify the actual software running on these systems. When possible, we encourage the use of formally verified software, like seL4 [23], which provides proofs of functional correctness, binary correctness, and that the API enables integrity and confidentiality. Additionally, recent work demonstrating the verification of complex systems software [27] and the development of lightweight verification methods suggests that the time is ripe to explore verifying satellite software.

Other assurance technologies like symbolic execution and model checking, automatic testing, test suite generation, and fuzzing are also promising and would be valuable for satellite software.

C. Fault Recovery

Faults are inevitable in satellite systems, due either to software bugs, hardware failures, or malicious attackers. A particularly critical challenge for satellite software is how to handle these faults and recover the system while preserving safety and availability.

Traditionally, the security community has considered that the best thing to do upon detecting a compromise was to crash the process or the system. The other common option, especially in operations, is to simply log the detection and do nothing. Unfortunately, neither of these are acceptable options for space systems, where the system needs to maintain operation in order to stay safe and be repaired, and where a ground operator may not be available sufficiently quickly to prevent irreversible damage.

One interesting alternative is to restart only some components of a system. This is both faster and less disruptive than rebooting or restarting the whole system. However, restarting individual components is also particularly challenging, as components must be sufficiently modular and state shared between components must be invalidated and reconstructed. In terms of modularity, the more isolation there is between components, the more straightforward restarting one component will be. Consider NASA's core Flight System (cFS), which has modular components, but runs them as separate threads in a single process [8]. In such a system, a fault in one component can easily cause corruption of shared state or state belonging to another module, which cannot be corrected with individual component restarts.

Handling stale state between components is a major challenge for partial restarts like this. This state may be knowledge about what state other components are in or resource handles like file descriptors. Such state is inherently application-specific and difficult to generally reconstruct. However, there is some prior work in this area [44] that attempts to identify failures due to lack of state and generate the appropriate calls to refresh that state.

The larger issue with simply restarting, either a single component or the whole system, is that it may be insufficient to mitigate the security issue. Bugs or vulnerabilities can often be repeatedly and frequently triggered by an adversary once identified. This becomes even more challenging for software supply chain attacks, where both the vulnerability and exploit code may be part of the system image and will continue to exist after restart. In such scenarios, a software update from the ground may be the only viable solution.

Another interesting approach to dealing with compromise is to provide simple-but-less-efficient versions of some functionality and switch to those when a compromise is detected. Alternatively, it may be possible to develop verifiable implementations that provide a "bounding box" on correct functionality. This could then be used to limit the impact of malicious components. These and other concepts are ripe for exploration.

D. Characterizing and Handling Radiation Faults

Truly radiation hardened devices are by-necessity built with large physical features, resulting in weak performance and high cost compared to non-radiation-hardened boards (see II). Ultimately, the goal of complete radiation hardening is unattainable—even terrestrial systems can see effects from radiation periodically [19]. As a result, the trend in hardware seems to be moving from devices that are truly radiation hardened to devices that are merely radiation-tolerant.

Overall, then, software should be expecting to cope with more radiation faults in the future, not less. This is a major challenge because software is already buggy and barely works correctly in terrestrial environments where radiation is not a concern. Coping with hardware failures with random characteristics is daunting at best. Significant further analysis of the impact of these faults on complex software systems is critical.

Of particular concern is how radiation faults impact security critical functionality and the security goals being provided. Most security techniques involve maintaining some invariants about expected behavior. However, these invariants will be enforced by a trusted computing base (TCB) that is subject to arbitrary radiation faults. How do these faults impact the invariants being enforced? Are there invariants that are impractical in space because we expect them to be regularly violated by radiation faults? As a concrete example, is process isolation a reasonable assumption when subject to radiation faults, or must we assume that process isolation may regularly be violated (i.e., unintended pages mapped) due to radiation faults?

We have begun some initial analysis on this topic, focusing on process isolation and simulating the impact of faults on it. In particular, we want processes to contain only the intended inter-process dataflows and remain independent in execution, except when there is an intended dependency. We then injected 960,000 faults within the CPU core of a RISC-V system (approximating the SiFive U54-MC complex), excluding memories which we assume to be ECC-protected, for a number of different software configurations. Unsurprisingly, systems with limited hardware isolation capabilities, such as the base FreeRTOS configuration, failed to uphold either goal. Memory accesses were physically addressed and unchecked. Inclusion of basic protections, such as a carefully coordinated watchdog timer, dramatically increased the detection rate by a factor of three, as the memory access violations are correlated with loss of liveness. Inclusion of an MPU, while coarse in its protections, was also extremely effective to constrain memory access. An MMU, managed using the seL4 microkernel, proved less effective than the MPU for preventing unauthorized dataflows, but seL4 did well at maintaining the independence of processes, with only 3 violations observed. The reduced effectiveness in memory protection in the MMU was shown to be the combined translation/enforcement role of the MMU. Single-bit upsets to the configuration, stored in page tables and the TLB, can remap memory arbitrarily.

To estimate the effectiveness of this approach, we chose to approximate the performance of the the U54-MC complex in the Microchip PolarFire SoC using neutron radiation characterization data from the Microchip PolarFire FPGA, fabricated on the same 28nm SONOS process [39]. With this data, we can estimate violations of our goals approximately once per 66 years of execution time executing at solar minimum in a geosynchronous orbit.

Future research needs to explore other security techniques and relevant invariants. Analysis is needed across all levels of fidelity and a wide variety of software systems and security functions.

E. Interoperability with Legacy Code

For new software for satellite systems, we encourage the use of memory safe languages, like Rust and Go, and secure software design principles, along similar lines to [9]. However, given the complexity of satellite systems and the number of systems being integrated, it is likely that new satellite systems will have to integrate with legacy code for the foreseeable future.

Integrating legacy software exists on a broad spectrum, from using legacy libraries in new code to supporting opaque, unmodified software on unmodified operating systems via virtualization. Below, we identify a few interesting and promising points on that spectrum and highlight challenges.

One possible point on that spectrum is automatically translating legacy components to new systems. Consider a legacy cFS application that one would like to use with a new, security-focused flight software system. It may be possible to automatically translate that cFS app into an equivalent application for the new system. For many legacy systems that rely on message passing for communication with the rest of the system, this seems a promising approach.

Another point on this spectrum would be to create adapters that can translate messages between new and legacy software. So instead of automatically converting a legacy cFS application, one would write an adapter to sit between cFS and the new system and pass messages back and forth. This may be promising for more complex scenarios involving many cFS applications. The major challenge for any such adapters is around security and methods to prevent spoofing messages on either side of the adapter.

A final interesting point on the spectrum of integrating legacy software involves virtualization. In particular, the legacy software runs in a virtualized environment with its required operating system and runtime. New software integrates with it using a custom emulated communication device. A major challenge here is achieving acceptable performance for the virtualized software.

VI. Conclusion

In this work we have explored challenges unique to satellite software, particularly the lack of human access, the importance of safety and availability, and dealing with radiation. We presented a threat model for this software and discussed how these challenges are dealt with today and what makes them complex. Finally, we present a vision for future satellite software and discuss research work needed to achieve that vision. We hope to inspire additional work in these areas and on the security of satellite software in general.

References

[1] Aerospace Corporation, "Sparta: Space attack research and tactic analysis," 2022. [Online]. Available: https://sparta.aerospace.org/

[2] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, "Solar winds hack: In-depth analysis and countermeasures," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2021, pp. 1–7.

[3] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 1362–1380.

[4] Y. Ashibani and Q. H. Mahmoud, "Cyber physical systems security: Analysis, challenges and solutions," *Computers & Security*, vol. 68, pp. 81–97, 2017.

[5] B. Bailey and B. Roeher, "Hacking an on-orbit satellite: An analysis of the cysat 2023 demo," 2023. [Online]. Available: https://medium.com/the-aerospace-corporation/hacking-an-on-orbit-satellite-an-analysis-of-the-cysat-2023-demo-ae241e5b8ee5

[6] K. Bedingfield, R. Leach, and M. Alexander, *NASA Reference Publication 1390: Spacecraft System Failures and Anomalies Attributed to the Natural Space Environment*, ser. NASA RP. National Aeronautics and Space Administration, Marshall Space Flight Center, 1996. [Online]. Available: https://ntrs.nasa.gov/api/citations/19960050463/downloads/19960050463.pdf

[7] C. Cimpanu, "Microsoft: 70 percent of all security bugs are memory safety issues," Feb 2019. [Online]. Available: https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/

[8] core Flight System Team, "core flight system," 2023. [Online]. Available: https://cfs.gsfc.nasa.gov/

[9] J. Curbo and G. Falco, "A research agenda for space flight software security," in *2023 IEEE 9th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. IEEE, 2023, pp. 68–77.

[10] B. Cyr, Y. Long, T. Sugawara, and K. Fu, "Position paper: Space system threat models must account for satellite sensor spoofing," *SpaceSec23, Feb*, 2023.

[11] G. Dvorsky, "Aging hubble space telescope in safe mode following computer glitch," 2021. [Online]. Available: https://gizmodo.com/aging-hubble-space-telescope-in-safe-mode-following-com-1847122644

[12] S. Erwin, "Espa satellites maturing as the preferred ride for small national security payloads," 2023. [Online]. Available: https://spacenews.com/espa-satellites-maturing-as-the-preferred-ride-for-small-national-security-payloads/

[13] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, "A systematic framework to generate invariants for anomaly detection in industrial control systems." in *Network and Distributed Systems Security Symposium (NDSS)*, 2019.

[14] S. Fitzsimmons, "Reliable software updates for on-orbit cubesat satellites," MS Thesis, 2012. [Online]. Available: https://digitalcommons.calpoly.edu/theses/804/

[15] A. Greenberg, "The untold story of notpetya, the most devastating cyberattack in history," *Wired, August*, vol. 22, 2018.

[16] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[17] Y. He and S. Chen, "Comparison of heavy-ion induced SEU for D- and TMR-flip-flop designs in 65-nm bulk CMOS technology," *Science China Information Sciences*, vol. 57, no. 10, pp. 1–7, Oct. 2014. [Online]. Available: https://doi.org/10.1007/s11432-014-5100-1

[18] D. M. Hiemstra, V. Kirischian, and J. Brelski, "Single event upset characterization of the zynq ultrascale+ mpsoc using proton irradiation," in *2017 IEEE Radiation Effects Data Workshop (REDW)*, 2017, pp. 1–4.

[19] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*. Ann Arbor Michigan: ACM, Jun. 2021, pp. 9–16. [Online]. Available: https://dl.acm.org/doi/10.1145/3458336.3465297

[20] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying compiler-automated software fault tolerance to multiple processor platforms," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 321–327, 2020.

[21] S. Jero, J. Furgala, R. Pan, P. K. Gadepalli, A. Clifford, B. Ye, R. Khazan, B. C. Ward, G. Parmer, and R. Skowyra, "Practical principle of least privilege for secure embedded systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 1–13.

[22] A. Keliris and M. Maniatakos, "Icsref: A framework for automated reverse engineering of industrial control systems binaries," in *Network and Distributed Systems Security Symposium (NDSS)*, 2019.

[23] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish *et al.*, "sel4: Formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 207–220.

[24] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar, "Utilizing dynamically coupled cores to form a resilient chip multiprocessor," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 317–326.

[25] L. Laursen, "Satellite signal jamming reaches new lows," 2023. [Online]. Available: https://spectrum.ieee.org/satellite-jamming

[26] H. Lefeuvre, V.-A. Bădoiu, Y. Chien, F. Huici, N. Dautenhahn, and P. Olivier, "Assessing the impact of interface vulnerabilities in compartmentalized software," *arXiv preprint arXiv:2212.12904*, 2022.

[27] S.-W. Li, X. Li, R. Gu, J. Nieh, and J. Z. Hui, "A secure and formally verified linux kvm hypervisor," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1782–1799.

[28] L. Mathews, "Viasat reveals how russian hackers knocked thousands of ukrainians offline," 2022. [Online]. Available: https://www.forbes.com/sites/leemathews/2022/03/31/viasat-reveals-how-russian-hackers-knocked-thousands-of-ukrainians-offline/

[29] D. McKee, Y. Giannaris, C. O. Perez, H. Shrobe, M. Payer, H. Okhravi, and N. Burow, "Preventing kernel hacks with hakc," in *Proceedings 2022 Network and Distributed System Security Symposium. NDSS*, vol. 22, 2022, pp. 1–17.

[30] Microsoft Security Response Center, "Guidance on microsoft signed drivers being used maliciously," 2022. [Online]. Available: https://msrc.microsoft.com/update-guide/vulnerability/ADV220005

[31] R. Morales-Ferre, P. Richter, E. Falletti, A. de la Fuente, and E. S. Lohan, "A survey on coping with intentional interference in satellite navigation for manned and unmanned aircraft," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 249–291, 2019.

[32] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What you corrupt is not what you crash: Challenges in fuzzing embedded devices." in *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[33] NIST, "Oscal: the open security controls assesment language," 2023. [Online]. Available: https://pages.nist.gov/OSCAL/

[34] NXP Semiconductors, "Layerscape® 1046a and 1026a processors," accessed on 01/12/2024. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/layerscape-processors/layerscape-1046a-and-1026a-processors:LS1046A

[35] J. Pavur and I. Martinovic, "The cyber-asat: on the impact of cyber weapons in outer space," in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900. IEEE, 2019, pp. 1–18.

[36] J. Pavur, D. Moser, V. Lenders, and I. Martinovic, "Secrets in the sky: on privacy and infrastructure security in dvb-s satellite broadband," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 277–284.

[37] R. Radhakrishnan, W. W. Edmonson, F. Afghah, R. M. Rodriguez-Osorio, F. Pinto, and S. C. Burleigh, "Survey of inter-satellite communication for small satellite systems: Physical layer to network layer view," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2442–2473, 2016.

[38] B. Rajulu, S. Dasiga, and N. R. Iyer, "Open source rtos implementation for on-board computer (obc) in studsat-2," in *2014 IEEE Aerospace Conference*. IEEE, 2014, pp. 1–13.

[39] N. Rezzak, J. Wang, S. Varela, P. Mok, and A. Cai, "PolarFire Neutron Testing Report," https://www.microsemi.com/document-portal/doc_view/1244460-polarfire-neutron-see-test-report, accessed: Jan 9, 2024.

[40] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[41] A. Schalk, L. Brodnik, and D. Brown, "Analysis of vulnerabilities in satellite software bus network architecture," in *MILCOM 2022-2022 IEEE Military Communications Conference (MILCOM)*. IEEE, 2022, pp. 350–355.

[42] D. Schmidt, K. Radke, S. Camtepe, E. Foo, and M. Ren, "A survey and analysis of the gnss spoofing threat and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–31, 2016.

[43] Y. Shen, G. Heiser, and K. Elphinstone, "Fault tolerance through redundant execution on cots multicores: Exploring trade-offs," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 188–200.

[44] J. Song, J. Wittrock, and G. Parmer, "Predictable, efficient system-level fault tolerance in c^ 3," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 21–32.

[45] I. Sünter, A. Slavinskis, U. Kvell, A. Vahter, H. Kuuste, M. Noorma, J. Kutt, R. Vendt, K. Tarbe, M. Pajusalu *et al.*, "Firmware updating systems for nanosatellites," *IEEE Aerospace and Electronic Systems Magazine*, vol. 31, no. 5, pp. 36–44, 2016.

[46] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 48–62.

[47] P. Tedeschi, S. Sciancalepore, and R. Di Pietro, "Satellite-based communications security: A survey of threats, solutions, and research challenges," *Computer Networks*, vol. 216, p. 109246, 2022.

[48] E. Tegler, "Someone in the middle east is leading aircraft astray by spoofing gps signals," 2023. [Online]. Available: https://www.forbes.com/sites/erictegler/2023/09/28/someone-in-the-middle-east-is-leading-aircraft-astray-by-spoofing-gps-signals/

[49] The Android Development Team, "A/b (seamless) system updates," 2023. [Online]. Available: https://source.android.com/docs/core/ota/ab

[50] The Chromium Project, "Memory safety," 2019. [Online]. Available: https://www.chromium.org/Home/chromium-security/memory-safety/

[51] The MITRE ATTACK Team, "Mitre — att&ck," 2015. [Online]. Available: https://attack.mitre.org/

[52] The MITRE CAPEC Team, "Capec: Common attack pattern enumeration and classification," 2023. [Online]. Available: https://capec.mitre.org/

[53] E. Tomur, U. Gülen, E. U. Soykan, M. A. Ersoy, F. Karakoç, L. Karaçay, and P. Çomak, "Sok: Investigation of security and functional safety in industrial iot," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2021, pp. 226–233.

[54] M. Torrieri, "How satellite imagery magnified ukraine to the world," 2022. [Online]. Available: https://interactive.satellitetoday.com/via/november-2022/how-satellite-imagery-magnified-ukraine-to-the-world/

[55] L. Tung, "Spacex: We've launched 32,000 linux computers into space for starlink internet," 2020. [Online]. Available: https://www.zdnet.com/article/spacex-weve-launched-32000-linux-computers-into-space-for-starlink-internet/

[56] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[57] B. C. Ward, R. Skowyra, S. Jero, N. Burow, H. Okhravi, H. Shrobe, and R. Khazan, "Security considerations for next-generation operating systems for cyber-physical systems," in *1st International Workshop on Next-Generation Operating Systems for Cyber-Physical Systems (NGOSCPS)*, 2019.

[58] R. N. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie *et al.*, "Cheri: A hybrid capability-system architecture for scalable software compartmentalization," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 20–37.

[59] D. Werner, "Hawkeye 360 detects gps interference in ukraine," 2022. [Online]. Available: https://spacenews.com/hawkeye-360-gps-ukr/

[60] J. Willbold, M. Schloegel, M. Vögele, M. Gerhardt, T. Holz, and A. Abbasi, "Space odyssey: An experimental software security analysis of satellites," in *IEEE Symposium on Security and Privacy*, 2023.

[61] C. Yan, H. Shin, C. Bolton, W. Xu, Y. Kim, and K. Fu, "Sok: A minimalist approach to formalizing analog sensor security," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 233–248.