Attack Surface Analysis for Spacecraft Flight Software

James Curbo and Gregory Falco

June 3, 2024

Attack Surface Analysis for Spacecraft Flight Software

James Curbo
Whiting School of Engineering
Johns Hopkins University
Baltimore, USA
jcurbo1@jhu.edu

Gregory Falco
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, USA
gfalco@cornell.edu

***Abstract*—We propose a method for enhancing cybersecurity in spacecraft operations by analyzing and reducing the attack surface of flight software. We advocate for reducing complexity in the software architecture and adopting more secure architectural principles to mitigate vulnerabilities and make spacecraft more resilient against cyber attacks. By utilizing a systematic approach, we scrutinize key areas, such as the real-time operating system (RTOS) and operating system abstraction layer (OSAL), and develop mitigations for issues we find. This study's findings suggest strategies for simplifying abstractions to make them more secure, addressing implementation issues, and providing supporting evidence for moving to a more resilient architectural approach.**

***Index Terms*—attack surface analysis, spacecraft flight software, cybersecurity for space systems, operating system abstraction layer, real-time operating system, software security engineering**

I. Introduction

The Real-Time Operating System (RTOS) plays a pivotal role in the operation of modern spacecraft, serving as the foundation upon which spacecraft software engineers develop mission flight software. It orchestrates the execution of various tasks, manages hardware resources, and ensures that time-critical operations are performed within stringent deadlines. The reliability and security of the RTOS is therefore crucial for the success of space missions, as any compromise could lead to catastrophic mission failures or significant data losses. As spacecraft technologies advance, the complexity of their RTOS environments has increased, introducing a broad spectrum of challenges for system developers and operators alike.

This burgeoning complexity in space RTOS presents a multifaceted problem. On one hand, it reflects the growing capabilities and functionalities of spacecraft, allowing for more sophisticated missions and scientific explorations. It escalates the potential attack surface, offering adversaries a wider array of vectors to exploit. The unique operating environment of space—characterized by remote operations, limited physical access for maintenance or updates, and extreme operational conditions—further amplifies these cybersecurity challenges. Traditional approaches to securing terrestrial systems often fall short when applied to the space domain, necessitating a more nuanced and tailored approach to understanding and mitigating risks.

Given the critical nature of spacecraft operations and the unique attributes of the space environment, it becomes imperative to adopt a specialized attack surface analysis approach. Such an approach must account for the distinct operational, technical, and environmental constraints of space systems. It should aim not only to identify potential vulnerabilities but also to understand the contextual significance of each in the broader scope of space mission operations. This demands a thorough examination of the RTOS and associated software layers, considering both their individual complexities and their interdependencies.

Our stance advocates for a strategic reduction in the attack surface through simplification and a focus on secure-by-design principles. By addressing the inherent complexities within the RTOS and the broader flight software stack, we can identify and mitigate potential vulnerabilities, enhancing the cyber resilience of spacecraft. This entails a careful balance between maintaining the functionalities for mission success and minimizing unnecessary complexities that may introduce security risks. Through this lens, we explore the current landscape of RTOS in space systems, analyze the challenges posed by their complexity, and propose methodologies for a comprehensive attack surface analysis tailored to the unique demands of space operations.

II. Background and Related Work

Understanding the multifaceted nature of spacecraft systems is crucial in increasing their cyber resilience. The complexity of these systems and their onboard flight software is ever-increasing, especially with spacecraft hosting high-powered computing payloads. This sea change is transforming spacecraft into highly distributed computing systems with varied hardware, operating systems, and application software. This shift from historical architectures centered on a single-board computer (SBC) is driven by the decreasing costs and size of compute and storage resources. The emergence of proliferated spacecraft constellations, comprising dozens to thousands of interconnected spacecraft, introduces additional distributed system complexities.

Such complexity not only complicates secure system design but also provides adversaries with opportunities to conceal their activities, deceive operators, and exploit hardware and software vulnerabilities for malicious purposes. The unique challenges of spaceflight, the space environment, and space system engineering further hinder the development of secure, resilient systems. This paper aims to address these cybersecurity implications at a foundational level, focusing on the operating systems that underpin spacecraft applications.

The interplay between system complexity and cybersecurity is critical; complexity does not inherently diminish resilience, but can obscure the full awareness of resilience properties and the understanding of risk. Therefore, spacecraft developers must thoroughly understand the attack surface of a system in order to effectively reduce the risk of cyber threats and ensure system states that are safe and secure. The following quote from Dr. Ron Ross, NIST Fellow, succinctly sums up the situation facing system operators.

> The real cyberwar is being fought on the field of complexity. It cannot be won with cybersecurity frameworks, tools, controls, assessments, zero trust concepts, or artificial intelligence alone. It will take a bare-knuckled, pound it out on the ground a yard at a time, systems and security engineering approach — applying rigorous design principles and architectures that minimize complexity and maximize assurance and trust. If you cede control of critical components such as operating systems to adversaries by failing to address complexity and assurance, they will use subversion to own the cyber battle space and turn your high-tech into no-tech. [1]

A. Complexity in Spacecraft Computing Systems

Modern spacecraft exhibit a high level of computational complexity. At the most basic level, a straightforward monolithic design incorporates a single-board, general purpose embedded microprocessor running an RTOS. However, this represents just the baseline for monolithic systems; many missions with lower risk profiles expand their core systems, transitioning to terrestrial-style computers equipped with operating systems familiar to any Linux system administrator. Spacecraft are growing their onboard computing capabilities in a distributed manner. Many designs now feature multiple computing subsystems beyond the primary flight computer, embedding sophisticated computing environments within essential subsystems such as radios, guidance systems, power management, and increasingly, completely separate hosted payloads. Some spacecraft have shifted towards a distributed system model, essentially comprising payloads supported by a minimal spacecraft bus [2]. Accompanying this surge in computing power are the interfaces, connections, and networks that link them together. These elements enable advanced functionality through distributed, networked computing systems—capabilities that were unavailable to earlier spacecraft platforms.

This multiplicity of computing systems provides a vast playground for adversaries to explore and exploit, and a tough job for cyber defenders seeking to monitor and defend such systems.

B. Unique Aspects of Space Systems

Adding to the computational complexity, space presents unique challenges that cause additional complexities for reliable computing and networking. Thummala, Curbo, Amir, *et al.* highlight these aspects in communication, high stakes, limited physical access, and constrained computing environments [3].

Space communications, while sharing constraints with terrestrial IT and ICS/SCADA systems, have developed distinctly. The vast distances and sporadic nature of contact cause architectures leaning towards Delay/Disruption Tolerant Networking (DTN) [4]. Data formats and protocols are space-specific, often governed by dedicated standards bodies [5]. The openness of space communications implies that any entity capable of observing a spacecraft might attempt communication or interception. Amateurs have exploited this openness for tracking [6] and even reviving defunct spacecraft [7], illustrating the potential for cyber attackers to target assets previously thought secure.

The stakes in space operations are increasingly high, with society's growing reliance on space assets. Losing telecommunications capabilities poses recognized risks [8]. The importance of satellite data for weather forecasting and its implications for national policy is gaining attention [9]. The proliferation of GPS jammers and spoofers, available for public purchase, affects air traffic control [10] and wide-area GPS interference impacts global aviation and other operations [11], highlighting the attractiveness of cyber means for adversaries aiming to disrupt or deny services.

The absence of physical access post-launch, traditionally a security advantage, becomes a challenge for cyber incident response, as physical intervention for mitigation and remediation is not workable. Malicious actors may even repurpose decommissioned, unmonitored satellites for offensive cyber operations [12].

Lastly, the constrained computing environment aboard spacecraft limits traditional cybersecurity approaches. The available computational resources, memory, and networking are insufficient for cybersecurity needs, prioritized instead for mission-critical tasks. The unique and often proprietary nature of spacecraft systems means general IT cybersecurity solutions are incompatible, highlighting the need for tailored approaches [13].

C. Mapping a Spacecraft's Attack Surface

Operators and cyber defenders must equip themselves with comprehensive knowledge of system vulnerabilities to counteract potential adversaries preemptively, because of the inherent complexity and unique challenges of space systems. This knowledge is pivotal not only for designing resilient systems but also for mitigating issues during development and fortifying production systems against cyber threats. The concept of an "attack surface" encompasses the various ways an attacker can access, influence, or communicate with a system [14]. A critical task for system developers is to understand this surface from both offensive and defensive perspectives, enabling the formulation of effective countermeasures.

Recent literature has outlined methodologies for conducting attack surface analysis specifically for space systems. The Space Attack Research & Tactic Analysis (SPARTA) framework emerges as a notable effort in this direction, offering a structured approach to categorize and document threats specific to spacecraft [15]. This framework complements existing threat analysis methodologies, such as Microsoft's STRIDE [16], the PASTA process [17], among others [18], adapting them to the unique exigencies of space systems.

Despite these advancements, detailed attack surface studies of spacecraft flight software and vehicles remain scarce in public discourse. While reports on cyber incidents, like the ViaSat attack [19] and the CYSAT demonstration [20], offer high-level insights, they fall short of providing the granular technical detail requisite for informing cyber resilience strategies or identifying specific architectural vulnerabilities in spacecraft flight software.

Conversely, substantial literature exists within similarly constrained domains employing embedded microprocessors or cyber-physical systems (CPS). Papp, Ma, and Buttyan [21] present a broad overview of threats and a taxonomy for attack categorization, reflecting the principles of SPARTA. Moreover, Easwaran, Chattopadhyay, and Bhasin [22] discuss considerations for real-time systems. The automotive industry, facing increased computing and networking demands, emphasizes secure design and threat understanding [23] [24], including autonomous vehicle challenges [25]. Similarly, the aviation sector is examining avionics and control system security [26] [27]. Among these, industrial control and SCADA systems have arguably advanced furthest in recognizing and addressing cybersecurity risks, as evidenced by guidelines from the U.S. Cybersecurity and Infrastructure Security Agency (CISA) [28], a dedicated MITRE ATT&CK framework [29], and many resources dedicated to enhancing system cyber resilience.

III. METHODOLOGY

A. Selection of Representative System

Identifying a representative flight software stack is crucial for effectively mapping the attack surface, evaluating analysis methodologies, and pinpointing necessary mitigation strategies. While many spacecraft systems leverage embedded platforms running Real-Time Operating Systems (RTOS), others employ more conventional computing platforms with operating systems like Linux. Despite operational differences between these systems, an analysis of an RTOS/embedded platform lays the groundwork for a broad understanding applicable to diverse flight software systems. The inherent software complexity of non-embedded platforms further underscores the importance of such an analysis, requiring a distinct investigation.

The criteria for selecting our representative system, not prioritized, include:

- Broad use across actual space missions to ensure the study's relevance and applicability to real-world space requirements and constraints.
- Public accessibility to facilitate in-depth analysis down to the source code level.
- Comprehensive coverage of spacecraft operations, from core hardware management to mission-specific tasks, to encompass a wide range of software use cases.

Based on these criteria, we chose NASA's *core Flight System* (cFS) [30] operating on the *Real-Time Executive for Multiprocessor Systems* (RTEMS) RTOS [31] for this study.

NASA's cFS, with a flight heritage dating back to the early 1990s across various platforms, is an open-source, extensible platform using a layered architecture. It supports multiple RTOSes beyond RTEMS, such as Wind River Software's vxWorks [32], Linux, and FreeRTOS [33]. RTEMS, selected for its open-source status and widespread use in the space sector and other embedded industries, aligns well with our selection criteria. The architecture of cFS is depicted in Fig. 1 [34, p. 54].

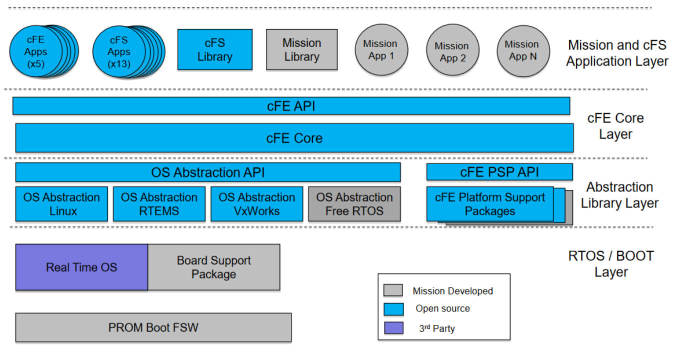

Fig. 1. Typical cFS Architecture

The selected stack comprises several layers, each contributing to the functionality and flexibility of the system:

- **Mission and Application Layer:** This layer encompasses applications provided by cFS developers and those custom-developed by mission owners for specific use cases, alongside supporting library code.
- **Core Flight Executive (cFE) Core Layer:** It delivers executive services and functions utilized by all applications, ensuring seamless operation across the board.
- **Abstraction Library Layer:** Housing the Operating System Abstraction Layer (OSAL) and cFE's Platform Support Library API, this layer facilitates the integration of OS-specific functionalities.
- **RTOS/Boot Layer:** Including the RTOS and board support package (BSP) necessary for running cFS on suitable hardware, this layer also accommodates any bootloaders required for system initialization.

The OSAL [35] plays a pivotal role in the cFS framework by providing seamless connectivity to the underlying operating system. It primarily comprises OS-specific wrapper code, which aligns the OSAL API with the corresponding OS

API (within the *src/os/* directory), and BSP code for each supported OS that configures device drivers and operating system modules as needed (within the *src/bsp/* directory).

Regarding OSAL's compatibility with RTEMS, it currently supports RTEMS 5.x, with advancements to accommodate RTEMS version 6.x observed in distinct development branches [36]. This study predominantly references RTEMS 5.3, the latest version at the time of writing, unless specified otherwise. RTEMS offers two APIs: the "classic" [37] and "POSIX" [38] APIs, with the OSAL utilizing the classic API for all interfacing. The RTEMS BSP (*src/bsp/pc-rtems/*) manages specific configurations and module inclusions during the RTEMS build process.

B. *Framework for Attack Surface Analysis*

The representative system previously discussed features a broad spectrum of functionalities across various software layers. Instead of dissecting each layer uniformly, this analysis will concentrate on the foundational elements of the software stack, specifically the Real-Time Operating System (RTOS) and the API infrastructure within the abstraction layer above the RTOS. Given its pivotal role in hardware control, software-hardware interactions, memory allocation, scheduling, and other core system functions, the RTOS stands as a critical focal point for initiating any attack surface analysis and bolstering cyber resilience.

We draw inspiration from Passmore and Ignatovich's approach to formal verification of financial algorithms [39]. While we are not dealing in formal verification of the flight software stack yet (we plan this for future work), there is value to consider lessons learned from Passmore and Ignatovich's work. In their paper, they consider a "stack of financial algorithms" used in the processing of orders for the financial industry that is similar in structure to many kinds of computing systems. Higher levels of the stack rely upon abstractions and mechanisms in lower levels of the stack, and the authors state that "one cannot properly reason about the possible behaviors of a system higher in the stack ... unless one has verified the relevant properties of the supporting subsystems that will be executing its intentions." [39, p. 27] This is a sound principle that guides our rationale in examining the RTOS; the rest of the flight software system completely depends on its behavior. Any attack on the RTOS can affect the overlying system, and manipulation of the RTOS may not be detectable at higher layers. Passmore and Ignatovich's lowest level mechanisms, order venues, deal in buy and sell orders for financial markets; this is directly analogous to fundamental RTOS management activities such as memory and task management. In both cases, these facilities simply must work the way developers designed them; without this verification, we have no firm foundation upon which to rest any analysis of the rest of the system. Once we have examined the lower layers of the stack, we can map weaknesses and vulnerabilities, change designs and put in place mitigations. Then we can turn our attention to higher levels of the stack, which will no doubt have their own issues.

To analyze specific attack surface aspects, we employ several methodologies. Dependency analysis explains the interrelations within system components, identifying those critical for overall functionality. With this information in hand, we can prioritize components for enhancing cyber resilience. Critical path analysis scrutinizes essential code paths, while data flow and interface analysis examine data movement and component communication, targeting potential vulnerabilities in data handling and interface use. Traditional vulnerability and code analysis seeks language usage flaws, improper function calls, and data management issues.

This study will prioritize dependency and critical path analysis as foundational investigation tools, guiding further examination with other methods. The focus will remain on the system's lowest layers—primarily the RTOS core, its board support package, and the cFS abstraction layer's OSAL and Platform Support Package (PSP). Analyzing the interactions between OSAL components, RTEMS managers, and device drivers will shed light on their connectivity, relationships, and standard configurations.

IV. Results and Discussion

A. *Complexity in the OSAL*

The Operating System Abstraction Layer (OSAL) in the core Flight System (cFS) enhances flexibility and separation of concerns, but introduces complexity that cyber attackers may exploit.

The OSAL is beneficial for cFS developers for various reasons. It enables the use of familiar operating systems from previous missions, supported by the OSAL, including RTEMS, vxWorks, and Linux. Embedded software developers widely use these operating systems, aligning with cFS's objectives. The Linux support within the OSAL, for instance, facilitates rapid prototyping, testing, training, or even operational deployment for missions without stringent real-time requirements. The OSAL standardizes the expression of cFS's operating system requirements through its API, simplifying system design and integration.

However, the OSAL approach has its drawbacks, impacting the system's attack surface and cyber resilience. Adhering to the principle of simplicity, which advocates for minimizing system complexity and redundant code, is crucial for enhancing cyber resilience [40, p. 102]. Although developers can make complex systems resilient against cyberattacks, simplification aids in identifying and mitigating potential vulnerabilities, preventing unsafe or insecure states. The added complexity requires more rigorous management and potentially the use of automated tools for ensuring system security and resilience.

The abstraction provided by the OSAL, through additional layers of code and indirection, can paradoxically undermine resilience. More code equates to more potential for errors, a concern amplified in abstraction layers or middleware because of their role in bridging disparate software paradigms. This requires careful design and vigilant management to handle the introduced complexity effectively. The increased codebase heightens the likelihood of vulnerabilities, requiring ongoing maintenance and adjustments in response to evolving external interfaces or dependencies. By obscuring interactions with the underlying system, the OSAL can also complicate the tuning of security controls and the detection of malicious activities within the upper layers of cFS.

The OSAL's implementation involves wrapping the target operating system's functionality with C source files and functions, as seen in RTEMS's case. This approach aims to present a uniform API to cFS, despite varying levels of operating system support for required functions. Wrapper implementations may include disparate amounts of code and sometimes additional logic to align the OSAL API with the operating system's capabilities. Such discrepancies introduce potential for bugs and vulnerabilities, underscoring the importance of diligent management and oversight.

B. RTEMS Execution Model

RTEMS exhibits a fairly standard architecture for a Real-Time Operating System (RTOS), characterized by its "real-time executive" that adopts a single process architecture [41]. In this model, all code shares a unified address space and is statically linked, except for the instances using RTEMS' dynamic loader. RTEMS allows task management through two distinct execution APIs: Classic or POSIX. However, because of its singular process architecture, RTEMS does not offer memory protection or security isolation among tasks, leading to a combined, statically linked binary image for the RTOS, API, and applications.

RTEMS supports dynamic loading of executable code and data (object files) [42], differing from conventional OS dynamic shared library support by integrating additional data into the singular address space as with statically linked binaries. While RTEMS' developers caution against employing dynamic loader functionality for real-time code—citing lack of protections and potential for misuse—this feature's unsafe usage can compromise real-time constraints and induce unpredictable behavior.

The absence of memory protection and task access control within RTEMS potentially allows malicious code to operate without restriction, enabling it to change memory and influence other tasks freely. Unlike operating systems designed with multiple users or access levels in mind, RTEMS prioritizes simplicity and efficiency, which, while aligning with its design objectives, poses challenges for cyber resilience. Developers must be vigilant of how malicious entities could infiltrate and persist within an RTOS's execution environment, particularly in space-bound systems, and devise strategies to counter such threats. Incidents involving vulnerabilities in FreeRTOS, which led to remote code execution, denial of service, and data leakage, underscore the importance of such considerations [43].

C. The RTEMS Shell

RTEMS incorporates a shell analogous to a conventional Unix shell, offering a range of similar commands [44]. While this feature is invaluable for development, debugging, and testing, its activation in operational spacecraft systems introduces significant cyberattack risks. The shell's capabilities include file system manipulation, system information queries, memory dumping and editing, and managing RTEMS' dynamic loader functions. Users can access the shell through a serial port or a network socket, contingent upon including RTEMS networking functionality in the build. RTEMS implements user and group-based access controls for command execution, featuring a "root" user akin to those in Unix systems. The OSAL activates the shell within its RTEMS Board Support Package (BSP) configuration, permitting the use of all commands [45, line 443].

Given RTEMS's architecture as a single address space RTOS lacking memory protection, exploits might change the shell's configuration for access or operate under "root" permissions. The shell's comprehensive command set offers attackers substantial "living off the land" capabilities, obviating the need for external tools for analysis or access—a technique increasingly used by cyber attackers [46].

D. RTEMS Board Support Package Configuration

RTEMS provides hardware-specific support and device drivers through board support packages. The OSAL's configuration of the RTEMS BSP, outlined in the *bsp/pc-rtems/src/bsp_start.c* file, is executed using RTEMS-defined parameters. Notably, the OSAL's default setup merits scrutiny from a cyber resilience perspective.

Configured to support four file systems—In-Memory File System (ImFS), DOS (FAT) file system, device file system (DevFS), and RTEMS File System (RFS)—ImFS serves as the default. However, the configuration does not disable any ImFS functionalities, though such restrictions are possible.

Attackers exploiting the file systems might alter data beneath an active task, leading to errors or system crashes. Vulnerabilities within unused file system driver code, remaining in memory, could serve as vectors for malicious exploits. Disabling non-essential file systems and curbing ImFS functionalities to prevent unsafe operations are advisable strategies for enhancing system security.

E. Memory Safety Considerations

The discourse on memory safety within the realm of spacecraft flight software, primarily developed in C, reveals a critical vector for potential cyber threats. As identified in our initial analysis, the traditional reliance on C introduces notable risks due to its lack of inherent memory safety features [47]. This susceptibility accentuates the broader issue of an expanded attack surface within space systems, necessitating a focused reevaluation of memory safety strategies to fortify cyber resilience.

F. Memory Safety Considerations

The discourse on memory safety within the realm of spacecraft flight software, primarily developed in C, reveals a critical vector for potential cyber threats. As identified in our initial analysis, the traditional reliance on C introduces notable risks because of its lack of inherent memory safety features [47]. This susceptibility stresses the broader issue of an expanded attack surface within space systems, causing a focused reevaluation of memory safety strategies to fortify cyber resilience.

1) Memory Safety and Attack Surface Expansion: The intrinsic memory safety risks associated with C—such as buffer overflows and unmanaged pointers—directly contribute to the expansion of a system's attack surface. These vulnerabilities offer attackers exploitable entry points, which can lead to unauthorized access, data corruption, or even system takeover. Understanding and mitigating these vulnerabilities is paramount for reducing the attack surface of RTOS-based space systems. Future work should include a targeted analysis of how attackers can exploit specific memory safety issues in space missions, identifying high-risk areas within the software stack that require immediate attention.

2) Balancing Language Choice with Operational Needs: While the adoption of memory-safe languages presents an appealing solution for reducing the attack surface, the transition from C must consider operational requirements and legacy system compatibility. Languages like Rust provide memory safety without sacrificing performance, making them attractive candidates for space software development. However, evaluating the implications of integrating these languages into existing ecosystems is crucial. Future efforts should focus on practical strategies for gradually incorporating memory-safe languages, possibly through mixed-language projects or by prioritizing their use in new mission-critical components.

3) Leveraging Tools for Memory Safety Assurance: Enhancing memory safety in C through the use of static and dynamic analysis tools represents an immediate step towards minimizing the attack surface. These tools can identify potential vulnerabilities during the development phase, allowing for their remediation before deployment. We propose that flight software developers perform an in-depth assessment of tool effectiveness specifically for their specific contexts, aiming to establish a toolkit recommendation that balances coverage, accuracy, and integration ease. This includes exploring automated tools that can support developers in adhering to secure coding practices and avoiding common pitfalls that lead to memory safety issues.

4) Guidelines and Best Practices for Secure Coding: Developing comprehensive guidelines and best practices for secure coding in the space sector can significantly impact the reduction of the attack surface. This effort should distill lessons learned from memory safety analyses and tool evaluations into actionable recommendations for developers. Emphasizing practices that prevent the introduction of memory safety vulnerabilities from the outset aligns with a proactive approach to attack surface management. Collaborating with standards bodies, such as the IEEE Space System Cybersecurity Working Group, to embed these practices within industry standards could further institutionalize memory safety as a foundational cybersecurity principle [48].

V. Future Work

The exploration undertaken in this study paves the way for a multitude of research avenues aimed at fortifying the cybersecurity landscape of spacecraft flight software. While we have laid the groundwork for understanding and mitigating the attack surface of RTOS-based systems in space missions, the ever-changing nature of cyber threats requires ongoing vigilance and innovation. Considering this, we propose several specific research directions to advance the state of cybersecurity in space systems. By pursuing these research directions, the scientific and engineering communities can continue to advance the cybersecurity of spacecraft systems, ensuring their resilience against the sophisticated cyber threats of today and tomorrow. This work not only contributes to the security of vital space missions, but also supports the broader goals of safe and sustainable space exploration and utilization.

A. Space-Specific Evaluation Framework for RTOS Cyber Resilience

Future research should aim to develop a comprehensive framework for evaluating the cyber resilience of various RTOS options available for space missions. This framework would consider factors such as memory safety, susceptibility to common vulnerabilities, and the support for secure coding practices. Comparative analyses conducted using this framework could guide system developers in selecting the most appropriate RTOS from a cybersecurity perspective.

B. Application of Formal Verification Methods to Space Software Systems

Given the critical nature of space missions, there is a pressing need to explore the feasibility and efficacy of applying formal verification methods to space software systems. This research direction would involve identifying suitable formal verification tools and techniques for space software and conducting case studies to assess their impact on detecting and mitigating vulnerabilities in RTOS-based systems.

C. Secure-by-Design Architectures for Next-Generation Spacecraft

As we advocate for a shift towards secure-by-design principles, future work should also focus on conceptualizing and developing next-generation spacecraft architectures that inherently prioritize cybersecurity. This includes investigating the integration of memory-safe programming languages, zero-trust security models adapted for space, and built-in security and isolation features. Research in this area could result in the creation of a prototype space-specific RTOS that embodies these principles. This would expand upon the work already being carried out in the IEEE Space Cybersecurity Working Group.

D. Cybersecurity Implications of Emerging Technologies in Space Systems

Integrating emerging technologies, such as artificial intelligence, machine learning, and blockchain into space systems offers promising benefits but also introduces new cybersecurity challenges. Future studies should aim to understand the cybersecurity implications of these technologies, assess their potential attack surfaces, and develop strategies to secure them against cyber threats.

E. Adversarial Testing in Space-Specific Testbeds

To validate the effectiveness of cybersecurity measures and to understand the practical implications of theoretical vulnerabilities, there is a need for adversarial testing within space-specific testbeds. Such testing should leverage findings from code analysis and threat modeling to simulate realistic attack scenarios. Developing offensive threat actor emulation capabilities for these testbeds could provide valuable insights into the resilience of space software systems against sophisticated cyber attacks.

VI. Conclusion

Throughout this study, we have delved into the intricacies of Real-Time Operating Systems (RTOS) in spacecraft operations, highlighting the critical role they play in mission success and the complexities they introduce. Our investigation into the attack surface of RTOS-based systems, particularly those used in space missions, underscores the urgent need for cybersecurity measures tailored to the unique challenges of the space environment. By focusing on the foundational layers of the RTOS and the abstraction layer, we have identified significant vulnerabilities that adversaries could exploit, potentially compromising mission integrity and data security.

The analysis presented reinforces the importance of adopting a secure-by-design philosophy in the development of spacecraft systems. Simplification of the RTOS architecture, alongside a conscientious effort to minimize unnecessary complexities, emerges as a key strategy in reducing the attack surface. Our exploration into memory safety and the utilization of memory-safe programming languages suggests a significant change that could significantly mitigate risks associated with common vulnerabilities in space systems.

Looking forward, the transition towards architectures that inherently prioritize cyber resilience stands as a necessary recommendation. This includes reevaluating the necessity of abstraction layers and considering the development of space-specific RTOS solutions that embed secure-by-design principles from the outset. Introducing advanced threat modeling and adversarial testing represents critical next steps in fortifying spacecraft against cyber threats that will provide a deeper understanding of adversary capabilities.

In conclusion, this study not only highlights the current cybersecurity challenges faced by spacecraft systems but also charts a course for future research and development efforts. By embracing a holistic approach to security, grounded in the unique operational realities of space missions, we can ensure the continued success and safety of space exploration endeavors. The journey towards more secure spacecraft systems is complex and ongoing, but it is essential for safeguarding the final frontier against growing cyber threats.

References

[1] R. Ross. "Ron ross quote on cyber resilience." (Nov. 2023), [Online]. Available: https://www.linkedin.com/feed/update/urn:li:activity:7147993287511982080/ (visited on 02/28/2024).

[2] Northrop Grumman. "ESPAStar," Northrop Grumman. (n.d.), [Online]. Available: https : www . northropgrumman . com / space / espastar (visited on 02/27/2024).

[3] R. Thummala, J. Curbo, Y. Amir, *et al.*, "Why is space cybersecurity unique?" Unpublished manuscript, Jan. 2024, Available upon request.

[4] "Delay/disruption tolerant networking (dtn)." (n.d.), [Online]. Available: https://datatracker.ietf.org/wg/dtn/about/ (visited on 02/27/2024).

[5] "CCSDS.org - the consultative committee for space data systems (CCSDS)." (n.d.), [Online]. Available: https://public.ccsds.org/default.aspx (visited on 02/27/2024).

[6] B. Tingley. "China's space plane apparently deployed 6 'mysterious wingmen' in orbit," Space.com. (Dec. 18, 2023), [Online]. Available: https://www.space.com/china-space-plane-depoyed-mystery-objects (visited on 02/27/2024).

[7] A. Klein, "Meet the amateur astronomer who found NASA's lost satellite in space," *Washington Post*, Oct. 27, 2021, ISSN: 0190-8286. [Online]. Available: https://www.washingtonpost.com/news/inspired-life/wp/2018/02/01/this-amateur-astronomer-found-a-satellite-lost-in-space/ (visited on 02/27/2024).

[8] G. Dvorsky. "What would occur if all of our satellites were... suddenly destroyed?" (Sep. 2015), [Online]. Available: http://satmagazine.com/story.php?number=1854194994 (visited on 02/27/2024).

[9] U. S. Government Accountability Office. "Environmental satellites: Launch delayed; NOAA faces key decisions on timing of future satellites — u.s. GAO." (n.d.), [Online]. Available: https://www.gao.gov/products/gao-16-143t (visited on 02/27/2024).

[10] "Truck driver has GPS jammer, accidentally jams newark airport," CNET. (n.d.), [Online]. Available: https://www.cnet.com/culture/truck-driver-has-gps-jammer-accidentally-jams-newark-airport/ (visited on 02/27/2024).

[11] "Russia behind spike in european GPS jamming, baltic general says - bloomberg." (n.d.), [Online]. Available: https://www.bloomberg.com/news/articles/2024-01-31/russia-behind-spike-in-european-gps-jamming-baltic-general-says?leadSource=reddit_wall (visited on 02/27/2024).

[12] "Hunting for space radio pirates on the US military fleet satcom satellites," rtl-sdr.com. (Mar. 3, 2023), [Online]. Available: https://www.rtl-sdr.com/hunting-for-space-radio-pirates-on-the-us-military-flt-satcom-satellites/ (visited on 02/27/2024).

[13] N. Tsamis, B. Bailey, and G. Falco, "Translating space cybersecurity policy into actionable guidance for space vehicles," in *ASCEND 2021*, _eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2021-4051, American Institute of Aeronautics and Astronautics, Nov. 3, 2021. DOI: 10.2514/6.2021-4051. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-4051 (visited on 05/29/2023).

[14] NIST Computer Security Resource Center. "Attack surface - glossary — CSRC." (n.d.), [Online]. Available: https://csrc.nist.gov/glossary/term/attack_surface (visited on 02/28/2024).

[15] The Aerospace Corporation. "Space Attack Research & Tactic Analysis (SPARTA)." (n.d.), [Online]. Available: https://sparta.aerospace.org/ (visited on 02/27/2023).

[16] L. Kohnfelder and P. Garg, "The threats to our products," *Microsoft Interface*, Apr. 1, 1999. [Online]. Available: https://shostack.org/files/microsoft/The-Threats-To-Our-Products.docx.

[17] M. Morana and T. Ucedavelez. "Risk analysis of banking malware attacks." (Jun. 10, 2011), [Online]. Available: https://www.slideshare.net/marco_morana/owasp-app-seceu2011version1 (visited on 02/28/2024).

[18] "Threat modeling: 12 available methods." (Dec. 2, 2018), [Online]. Available: https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/ (visited on 02/28/2024).

[19] N. Boschetti, N. G. Gordon, and G. Falco, "Space cybersecurity lessons learned from the ViaSat cyberattack," in *ASCEND 2022*, ser. ASCEND, American Institute of Aeronautics and Astronautics, Oct. 13, 2022. DOI: 10.2514/6.2022-4380. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2022-4380 (visited on 12/07/2023).

[20] B. Bailey and B. Roeher. "Hacking an on-orbit satellite: An analysis of the CYSAT 2023 demo," Aerospace TechBlog. (May 25, 2023), [Online]. Available: https://medium.com/the-aerospace-corporation/hacking-an-on-orbit-satellite-an-analysis-of-the-cysat-2023-demo-ae241e5b8ee5 (visited on 05/29/2023).

[21] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, Jul. 2015, pp. 145–152. DOI: 10.1109/PST.2015.7232966. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7232966 (visited on 02/09/2024).

[22] A. Easwaran, A. Chattopadhyay, and S. Bhasin, "A systematic security analysis of real-time cyber-physical systems," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, ISSN: 2153-697X, Jan. 2017, pp. 206–213. DOI: 10.1109/ASPDAC.2017.7858321. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7858321 (visited on 01/27/2024).

[23] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, *et al.*, "Security challenges in automotive hardware/software architecture design," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, ISSN: 1530-1591, Mar. 2013, pp. 458–463. DOI: 10.7873/DATE.2013.102. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6513548 (visited on 02/07/2024).

[24] J. Edwards, A. Kashani, and G. Iyer, "Evaluation of software vulnerabilities in vehicle electronic control units," in *2017 IEEE Cybersecurity Development (SecDev)*, Sep. 2017, pp. 83–84. DOI: 10.1109/SecDev.2017.26. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8077811 (visited on 01/24/2024).

[25] K. Zhang and A. Olmsted, "Examining autonomous vehicle operating systems vulnerabilities using a cyber-physical approach," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, Sep. 2021, pp. 976–981. DOI: 10.1109/ITSC48978.2021.9564848. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9564848 (visited on 01/27/2024).

[26] T. Kiesling, M. Krempel, J. Niederl, and J. Ziegler, "A model-based approach for aviation cyber security risk assessment," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug. 2016, pp. 517–525. DOI: 10.1109/ARES.2016.63. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7784614 (visited on 02/28/2024).

[27] E. Habler, R. Bitton, and A. Shabtai, "Assessing aircraft security: A comprehensive survey and methodology for evaluation," *ACM Computing Surveys*, vol. 56, no. 4, 96:1–96:40, Nov. 10, 2023, ISSN: 0360-0300. DOI: 10.1145/3610772. [Online]. Available: https://dl.acm.org/doi/10.1145/3610772 (visited on 02/28/2024).

[28] US CISA. "Seven steps to effectively defend industrial control systems_s508c.pdf." (n.d.), [Online]. Available: https://www.cisa.gov/sites/default/files/documents/Seven%20Steps%20to%20Effectively%20Defend%20Industrial%20Control%20Systems_S508C.pdf (visited on 02/28/2024).

[29] The MITRE Corporation. "Techniques - ICS — MITRE ATT&CK®." (n.d.), [Online]. Available: https://attack.mitre.org/techniques/ics/ (visited on 02/28/2024).

[30] NASA Goddard Space Flight Center. "Core flight system." (n.d.), [Online]. Available: https://cfs.gsfc.nasa.gov/ (visited on 02/24/2024).

[31] The RTEMS Project. "RTEMS real time operating system (RTOS) — real-time and real free RTOS." (n.d.), [Online]. Available: https://www.rtems.org/ (visited on 02/24/2024).

[32] Wind River Systems. "VxWorks — industry leading RTOS for embedded systems." (n.d.), [Online]. Available: https://www.windriver.com/products/vxworks (visited on 02/24/2024).

[33] The FreeRTOS Project. "FreeRTOS - market leading RTOS (real time operating system) for embedded systems with internet of things extensions," FreeRTOS.

(n.d.), [Online]. Available: https://www.freertos.org/index.html (visited on 02/24/2024).

[34] NASA Goddard Space Flight Center. "NASA core flight system (cFS) background and overview." (n.d.), [Online]. Available: https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf (visited on 02/24/2024).

[35] NASA Goddard Space Flight Center. "Nasa/osal: The core flight system (cFS) operating system abstraction layer (OSAL)." (n.d.), [Online]. Available: https://github.com/nasa/osal (visited on 02/25/2024).

[36] A. Cudmore, *Alanc98/rtems-cfs-demo*, original-date: 2021-12-24T18:30:42Z, Jan. 21, 2024. [Online]. Available: https://github.com/alanc98/rtems-cfs-demo (visited on 02/25/2024).

[37] RTEMS Documentation Project. "RTEMS classic API guide (5.3). — RTEMS classic API guide 5.3 (10th february 2023) documentation." (n.d.), [Online]. Available: https://docs.rtems.org/releases/rtems-5.3/c-user/index.html (visited on 02/25/2024).

[38] RTEMS Documentation Project. "RTEMS POSIX API guide (5.3). — RTEMS POSIX API guide 5.3 (10th february 2023) documentation." (n.d.), [Online]. Available: https://docs.rtems.org/releases/rtems-5.3/posix-users/index.html (visited on 02/25/2024).

[39] G. O. Passmore and D. Ignatovich, "Formal verification of financial algorithms," in *Automated Deduction – CADE 26*, L. de Moura, Ed., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 26–41, ISBN: 978-3-319-63046-5. DOI: 10.1007/978-3-319-63046-5_3.

[40] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing cyber-resilient systems: A systems security engineering approach," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-160v2r1, Dec. 7, 2021, NIST SP 800–160v2r1. DOI: 10.6028/NIST.SP.800-160v2r1. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2r1.pdf (visited on 01/17/2023).

[41] The RTEMS Project. "1. introduction — RTEMS user manual 5.3 (10th february 2023) documentation." (n.d.), [Online]. Available: https://docs.rtems.org/releases/rtems-5.3/user/overview/index.html#real-time-executive (visited on 02/29/2024).

[42] The RTEMS Project. "8.7. dynamic loader — RTEMS user manual 5.3 (10th february 2023) documentation." (n.d.), [Online]. Available: https://docs.rtems.org/releases/rtems-5.3/user/exe/loader.html (visited on 02/29/2024).

[43] E. Kovacs. "FreeRTOS vulnerabilities expose many systems to attacks," SecurityWeek. (Oct. 19, 2018), [Online]. Available: https://www.securityweek.com/freertos-vulnerabilities-expose-many-systems-attacks/ (visited on 02/29/2024).

[44] RTEMS Documentation Project. "RTEMS shell guide (5.3). — RTEMS shell guide 5.3 (10th february 2023) documentation." (n.d.), [Online]. Available: https://docs.rtems.org/releases/rtems-5.3/shell/index.html (visited on 02/25/2024).

[45] NASA Goddard Space Flight Center. "Osal/src/bsp/pc-rtems/src/bsp_start.c." (n.d.), [Online]. Available: https://github.com/nasa/osal/blob/53550cafb718f5f608da4240a8525f75ff5bb612/src/bsp/pc-rtems/src/bsp_start.c#L65 (visited on 02/25/2024).

[46] Crowdstrike. "What are living off the land (LOTL) attacks?" crowdstrike.com. (n.d.), [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/living-off-the-land-attacks-lotl/ (visited on 02/25/2024).

[47] J. Curbo and G. Falco, "Memory safety in space," Unpublished manuscript, Mar. 2024, Available by request.

[48] IEEE Standards Association. "P3349 - space system cybersecurity working group - home." (Mar. 2024), [Online]. Available: https://sagroups.ieee.org/3349/ (visited on 02/28/2024).