SECURING SPACE SYSTEMS: ENGINEERING SOLUTIONS ACROSS THE LIFECYCLE

Zain A. H. Hammadeh *, Daniel Lüdtke *, Michael Felderer *†
* German Aerospace Center (DLR), Institute of Software Technology, Germany
† University of Cologne, Germany

Abstract

The growing commercialization, militarization, and technological complexity of space missions have increased the exposure of space systems to cyber threats. Ensuring cybersecurity across the lifecycle of space software is essential to preserve mission integrity, continuity, and trust. This paper presents a comprehensive engineering approach to securing space systems, emphasizing the integration of cyber resiliency principles from design to decommissioning. We examine preventive measures, including threat modeling, secure design, and testing; detection strategies leveraging host- and network-based intrusion detection systems; responsive mechanisms for containment and mitigation; recovery procedures to restore system functionality; and resilience evolution processes for continuous learning and improvement. We highlight how lifecycle-driven cybersecurity enables robust, resilient, and trustworthy space systems.

Keywords
cyber resilience software; cybersecurity; space systems

1. INTRODUCTION

The rapid expansion of space exploration, coupled with the increasing commercialization and militarization of space technologies, has made the security of space assets a critical concern. Space systems—including satellites, ground stations, and user terminals—face an evolving array of cyber threats that can compromise mission success, data integrity, and operational continuity. Historically, spacecraft were considered inherently secure due to their physical isolation; direct access to on-board systems was largely infeasible. However, this perception is no longer valid. Attackers now exploit sophisticated techniques such as spoofing, jamming, and manipulation of onboard sensors and communication channels. The growing diversity of space missions, spanning civilian, scientific, and military applications, has further expanded the attack surface and introduced novel vulnerabilities [1].

Modern satellite architectures, including CubeSats, increasingly rely on commercial off-the-shelf (COTS) hardware and third-party software to meet the growing demand for onboard computation and payload processing. Platforms are envisioned to host third-party applications, effectively functioning as cloud nodes in space [2]. General-purpose operating systems, such as Linux, are deployed to support advanced applications, including artificial intelligence (AI) and real-time data processing. While these technologies enhance mission capabilities, they also amplify cybersecurity risks. Supply chain attacks, vulnerabilities in open-source libraries, and potential backdoors in precompiled binaries present significant challenges to mission integrity and continuity. Addressing these threats requires integrating cybersecurity as a fundamental design principle rather than as a post-development add-on. FIG 1 illustrates how cybersecurity activities can be systematically integrated into the software V-model lifecycle for space systems. On the left side of the V, security considerations are embedded during requirements definition, architectural design, and detailed design, ensuring that prevention and resilience requirements are explicitly captured. On the right side, verification and validation activities incorporate security testing, including vulnerability assessments, intrusion simulations, and recovery drills. During Operation & Maintenance phase, detection, response, and recovery mechanisms are validated in realistic mission scenarios.

Early-stage activities such as threat modeling, risk assessment, and rigorous evaluation of third-party software are essential. These processes involve both static and dynamic testing at the source and binary levels to detect vulnerabilities, malicious behaviors, and design flaws. Beyond preventive measures, resilience is crucial: space systems must maintain essential functionality under attack. Achieving this requires redundancy, fail-over mechanisms, and rapid recovery protocols to ensure mission continuity despite cyber incidents.

The importance of lifecycle-driven cybersecurity for space systems has been highlighted in numerous studies and real-world events. For instance, the Viasat KA-SAT network cyberattack in 2022 demonstrated the

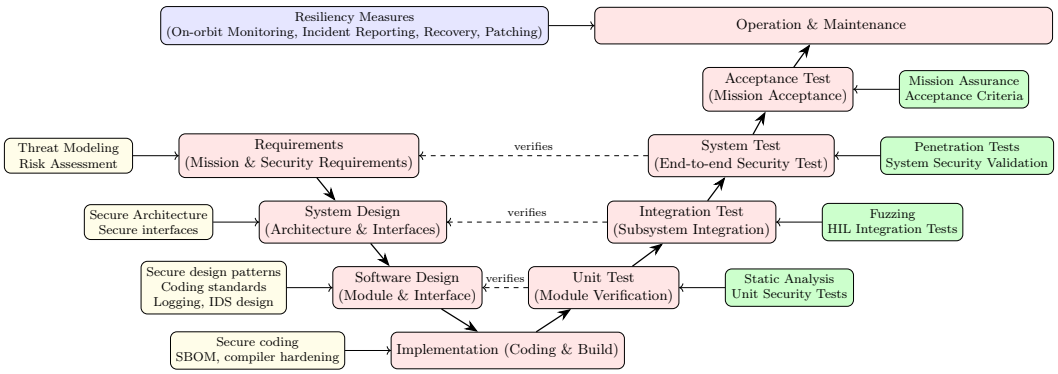

FIG 1. Integrating the Cybersecurity engineering process into the V-model for Space Systems.

operational impact of space-targeted cyber threats [3]. Willbold et al. [4] analyzed small satellite vulnerabilities using fuzzing techniques, while OrbitalShield [5] and Hammadeh et al. [6] emphasized the need for holistic, lifecycle-oriented security frameworks, particularly in the context of COTS-based missions.

While traditional cybersecurity standards were developed for general IT environments, recent efforts have produced space-specific guidance. Between 2022 and 2025, multiple international standards and best practices were released to support space companies in embedding security into all phases of mission development, from concept to decommissioning. Notable examples include ISO/TS 20517:2024 for lifecycle cybersecurity management [7], CCSDS reports on secure communication protocols [8], IEEE standards P3349 and P3536 [9, 10], and the ECSS-E-ST-80C security standard [11]. NASA has also published a Space Security Best Practices Guide [12]. Worth mentioning here are the technical guidelines for space (Part 1 [13]) and ground (Part 2 [14]) segments that are published by the German Federal Office for Information Security (BSI). While this paper does not prescribe adherence to a particular standard, these frameworks inform the engineering practices and solutions discussed herein.

This paper focuses on engineering solutions for achieving cyber-resilient space software. Section 2 presents the concept of cyber resiliency and its realization across the software lifecycle. Sections 3 through 5 discuss practical methods for prevention, detection, response, recovery, and evolution. Section 6 concludes the paper and summarizes the key findings.

2. CYBER RESILIENCY

Building on the foundation established in the introduction, ensuring the security of space software requires more than preventive measures alone. Modern spacecraft rely on complex software ecosystems [15], integrating diverse tasks across multiple embedded systems with varying levels of criticality. These mixed-criticality environments often include both proprietary and third-party software components, including COTS modules, which can introduce vulnerabilities if not carefully managed [16, 17].

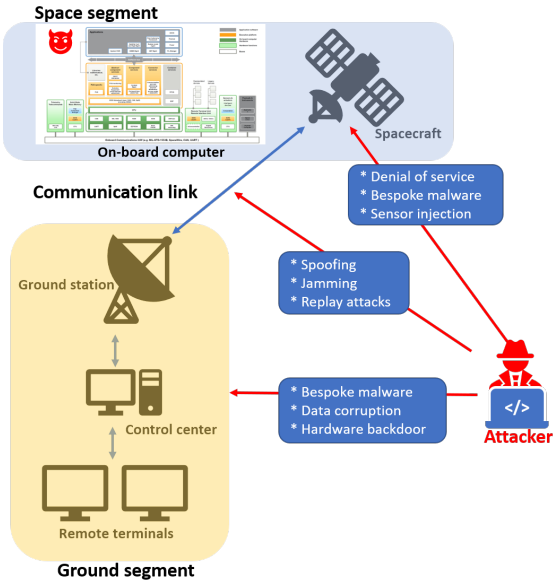

FIG 2. Different space infrastructure segments may be subject to different attacks.

Cyber resiliency in space systems encompasses the capacity to anticipate, withstand, and recover from cyber incidents while maintaining mission-critical functions. As illustrated in FIG 2, the primary targets include the ground segment, communication links, and the space segment itself. Each segment poses its own distinct security challenges: ground stations and mission control centers are critical hubs for satellite command and monitoring; communication links must resist spoofing, replay attacks, and jamming; while the spacecraft itself must operate reliably even when hosting third-party applications or executing on potentially vulnerable COTS hardware. Achieving cyber resiliency requires a lifecycle-driven approach, incorporating the principles of prevention, detection, response, recovery, and evolution. Preventive measures, implemented during system design and development, aim to reduce the attack surface through secure coding practices, threat modeling, and component hardening. Detection mechanisms, such as host-based, network-based, and hybrid intrusion detection systems (IDS), continuously monitor both software behavior and network traffic to identify

anomalies in real-time [18, 19]. Once an incident is identified, response strategies—ranging from automated on-board interventions to ground-assisted decisions—seek to contain the threat while preserving operational continuity [20, 21]. Recovery processes then restore full functionality, leveraging redundancy, rollback mechanisms, and verified backups to resume nominal operations. Finally, evolution integrates lessons learned into the system design, updates detection rules, and adapts operational procedures, thereby increasing resilience over the mission's lifespan.

This comprehensive perspective on cyber resiliency can be represented as a continuous feedback loop across the software lifecycle, as shown in FIG 3. In each lifecycle phase—requirements, design, implementation, testing, deployment, and operation—specific activities contribute to resilience. For instance, security and resiliency requirements identified during the concept phase are implemented through secure design patterns and runtime monitoring mechanisms. During operation, anomaly-based IDS models and telemetry monitoring support real-time threat detection, while automated recovery and redundancy measures ensure continuity of critical functions. Lessons learned from incidents and operational anomalies are subsequently fed back into design and maintenance, improving defenses and enabling adaptive system evolution.

TAB 1 maps these practices to the corresponding lifecycle phases, illustrating how preventive, detective, responsive, and restorative measures, combined with continuous evolution, collectively strengthen the cyber posture of space software. Cyber resiliency must be systematically integrated into systems engineering practices across the entire lifecycle, ensuring that security is treated with the same rigor as safety and reliability, and embedded as an intrinsic attribute of every mission. In the following section we elaborate on each principle of the cyber resiliency.

3. PREVENTION

Prevention represents the first line of defense in the cyber-resilient space software lifecycle, embodying the secure-by-design philosophy. Activities in this phase encompass threat modeling, risk assessment, requirements definition, vulnerability analysis, testing, and verification. The goal is to minimize potential vulnerabilities before the system is deployed in orbit, reducing the likelihood and impact of cyberattacks.

Practical preventive measures include isolating software components using containers [22,23] and employing secure kernels such as seL4 [24], which provide strong guarantees of memory safety and task isolation. Threat modeling is a foundational activity, identifying potential attack vectors and mapping them to mitigation strategies. Typical threats include spoofing—where attackers forge telecommands or telemetry to mislead the system [8,25]—and jamming, which denies communication by injecting noise into the uplink or downlink channels. Cyberattacks can target both data and system integrity, leveraging malware, legacy protocol vulnerabilities, or the injection of corrupted commands. The severity of such attacks varies, from temporary service disruptions to permanent loss of control over a satellite constellation, such as ransomware [26].

Across the software lifecycle, threat modeling is applied at each phase, guided by reference frameworks such as STRIDE [27], Common Weakness Enumeration (CWE)[1], OWASP [28], and MITRE ATT&CK for Space [29]. TAB 2 maps software lifecycle phases to relevant threat modeling activities. During requirements definition, mission functions are mapped to critical assets and associated security objectives. In the design phase, data flow diagrams (DFD) and trust boundaries are created to model potential attack surfaces. Implementation integrates secure coding practices, static analysis, and continuous monitoring, while integration and testing leverage fuzzing, fault injection, and automated CI/CD checks. During deployment and operations, threats are continuously monitored, and over-the-air (OAT) updates are secured against malicious modification. Finally, during decommissioning, access is disabled, and sensitive data is securely erased.

Malicious code insertion, dependency confusion, and unpatched vulnerabilities in external components represent significant risks for mission assurance. Unlike traditional ground-based systems, satellites often cannot be rapidly patched or physically serviced, making preemptive supply chain assurance essential. One effective mitigation strategy is the adoption of a Software Bill of Materials (SBOM) [30]. An SBOM provides a structured inventory of all software components, including libraries, dependencies, and their versions. Maintaining an SBOM throughout the lifecycle enables proactive vulnerability tracking, facilitates incident response, and increases trust in the software baseline. In addition to SBOMs, rigorous supplier vetting and certification, cryptographic provenance verification of binaries, and the enforcement of secure update mechanisms are critical.

To further enhance supply chain resilience, spacecraft software should undergo both static and dynamic analysis during integration, regardless of whether the source code is available. Binary-level scanning [31] can reveal hidden vulnerabilities or malicious behavior in third-party software, while runtime monitoring under simulated mission conditions can identify unexpected interactions. By incorporating these practices, mission teams can significantly reduce the attack surface introduced by external dependencies.

Attack-Fault Trees (AFTs) further facilitate joint analysis of safety and security in cyber-physical systems. By combining dynamic fault trees with attack trees, AFTs capture the interdependencies of software and hardware vulnerabilities across multiple abstraction layers [32]. Automated tools can generate comprehensive AFTs, enabling model-checking and scenario validation for complex systems.

[1] https://cwe.mitre.org/

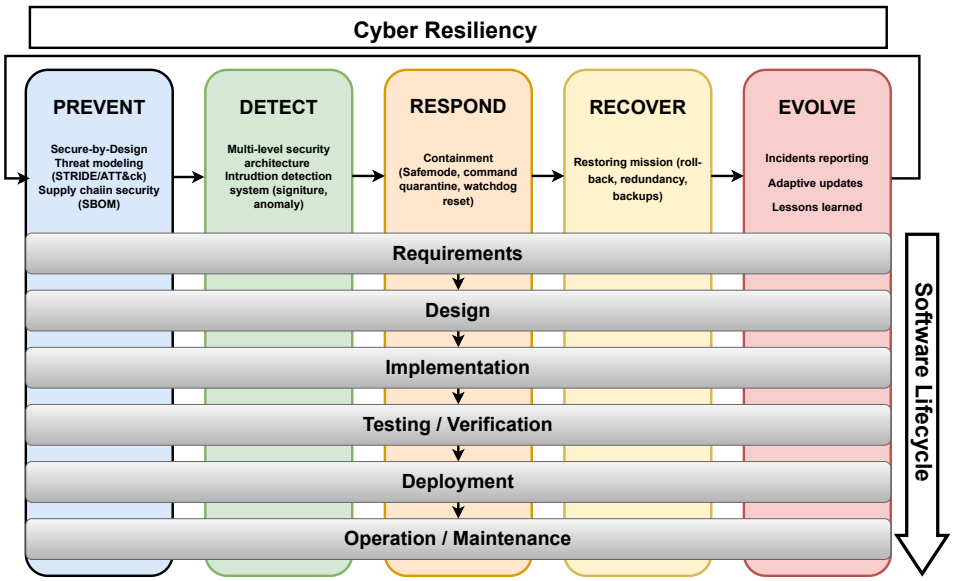

FIG 3. Cyber resiliency across the software lifecycle.

Testing and verification complement preventive measures. Software-in-the-loop (SIL) and hardware-in-the-loop (HIL) simulations allow validation of flight software and hardware interactions under normal and adversarial conditions. Penetration testing, both on the ground segment and in the supply chain, simulates attacker behavior to identify potential weaknesses. Formal verification and static analysis tools detect memory safety issues, verify authentication properties, and ensure that cryptographic operations are implemented correctly [31, 33]. Operational security drills, combined with digital twins [34, 35] and automated testing frameworks, ensure that preventive measures are robust and resilient against evolving threats.

4. DETECTION, RESPONSE AND RECOVERY

4.1. Detection

Detection is a critical element of cyber resiliency, enabling timely identification of attacks and anomalous behavior in both software and network operations. Efficient IDS form the backbone of detection strategies, continuously monitoring system behavior and network traffic to identify potential threats in real time [18]. In space systems, these mechanisms must operate under stringent resource constraints, requiring low-latency, lightweight solutions that preserve essential functionality while maintaining high reliability.

IDS solutions can be classified by their data sources and deployment locations:

- Host-based IDS (HIDS) monitor individual nodes, collecting metrics such as memory usage, execution times, and system calls to detect suspicious activity.
- Network-based IDS (NIDS) focus on traffic between nodes, inspecting uplinks, downlinks, and internal communications for anomalous patterns.
- Hybrid approaches combine both host and network monitoring, as exemplified by distributed IDS (DIDS), which correlate data across multiple nodes for comprehensive situational awareness.

Two principal methodologies underpin IDS design: knowledge-based and behavior-based detection. Knowledge-based systems rely on signatures or rules derived from known attacks, providing accurate detection for previously observed threats while maintaining low false-positive rates. Their limitation lies in their inability to detect zero-day attacks. In contrast, behavioral or anomaly-based detection methods identify deviations from established baselines, enabling the detection of previously unknown threats, although with a higher likelihood of false positives [19]. TAB 3 illustrates a comparison between signature IDS and anomaly IDS from a space software perspective. Explainable machine learning approaches are more and more proposed to improve onboard anomaly detection, reducing false positives and enhancing operator trust [36].

Integrating IDS throughout the software lifecycle ensures traceability from requirements to operations. Security requirements derived from threat modeling—such as detecting spoofed telecommands or unsigned updates—are directly mapped to detection mechanisms. For example, signature-based IDS may monitor for unauthorized telecommands or telemetry spoofing, while runtime monitoring detects abnormal memory access patterns that could indicate privilege escalation. Formal requirements, such as those recommended by NASA and ECSS, specify the logging, alerting, and telemetry behavior expected from the IDS, ensuring that detection is actionable and verifiable.

IDS integration occurs at multiple levels. Onboard IDS operate close to critical processes on the onboard computer (OBC), detecting low-level attacks such

Lifecycle Phase	Prevent (avoid vulnerabilities)	Detect (spot issues early)	Respond (contain incidents)	Recover (restore functionality)	Evolve (improve for future)
Requirements	Define security & resiliency requirements (e.g., command auth; update integrity)	Define logging/-monitoring requirements	Define incident handling procedures	Define recovery plans (e.g., rollback within X hours)	Capture lessons from prior missions; feed into requirements
Design	Apply secure design patterns; minimize attack surface	Design for observability (telemetry hooks; IDS interfaces)	Design safe modes & isolation mechanisms	Design redundancy (dual OBC images; watchdog resets)	Build adaptability (modular updates; re-trainable IDS models)
Implementation	Secure coding; static analysis; memory safety	Insert debug hooks; runtime checks	Implement containment (process isolation; sandboxing)	Implement rollback features & error-handling	Document design trade-offs for future iteration
Testing / Verification	Fuzzing; penetration testing; supply chain checks	Test anomaly detection & IDS alerts	Run red-team drills & incident playbooks	Test rollback & failover procedures	Capture test metrics & refine test cases for next mission
Deployment	Validate secure config; cryptographic keys; access controls	Verify IDS and logging are active in ops environment	Prepare incident response playbooks for operators	Deploy recovery tools (rollback image; reboot scripts)	Update deployment pipeline with lessons learned
Operation (In-Orbit)	Continuous monitoring; patching to prevent recurrence	Real-time anomaly detection (telemetry; network)	Contain threats via safe mode; command filtering	Restore functionality (restart services; upload clean software)	Feed ops incidents into evolving defenses (IDS retraining; updated policies)

TAB 1. Security and resilience practices mapped across lifecycle phases.

as buffer overflows, replay attacks, and telemetry spoofing. These systems leverage runtime monitors, lightweight anomaly detection algorithms, and telecommand validators, often embedded as security tasks within real-time operating systems such as RTEMS. Ground-assisted IDS provide oversight, enabling operators to validate suspicious commands, analyze telemetry for patterns, and support coordinated incident response. Mission-aware adaptive detection combines data from onboard and ground sensors to inform dynamic reconfiguration and operational decisions, preserving mission-critical functions even under attack.

The effectiveness of detection mechanisms can be evaluated through metrics such as mean time to detect ($MTTD$), false positive and false negative rates, as well as containment success rates. These metrics quantify the timeliness, accuracy, and operational impact of detection, informing improvements and tuning throughout the mission lifecycle.

4.2. Response

Detection alone is insufficient to guarantee the safety and security of space systems; appropriate responses must be executed to mitigate the effects of cyberattacks. Intrusion response systems (IRS) are designed to manage detected threats, contain malicious activity, and minimize damage while maintaining essential functionality. In satellites, simple strategies such as switching to safe mode and transmitting telemetry to the ground provide immediate containment, but more autonomous and adaptive responses are increasingly necessary to handle sophisticated threats, including AI-powered attacks, supply chain attacks, advanced persistent threats (APTs) [38]. However, the security priorities in space systems differ from typical terrestrial environments. Whereas conventional IT systems prioritize confidentiality, integrity, and availability (CIA) [39], satellites invert this hierarchy, placing safety and availability above other concerns [40]. Any mitigation strategy that risks shutting down a satellite or disrupting its primary functionality is unacceptable. Therefore, responses must be generic, lightweight, and fail-operational, ensuring that critical systems remain functional even while isolating compromised components [17, 20]. Reconfiguration-based strategies, commonly employed in fault-tolerant space systems, are well-suited as intrusion responses. By

Lifecycle Phase	Software Focus	Threat Modeling Activity
Requirements	Define mission functions (telecommanding; telemetry; payload tasking; data downlink)	Identify assets (flight software; communication protocols; payload data); define security objectives
Design	Architecture of OBC flight software, communication stacks, ground software	Create DFD models (Threagile YAML; Pytm Python); define trust boundaries (space ↔ ground; operator ↔ payload)
Implementation	OBC software and ground software	Apply secure coding practices; perform static analysis; integrate threat model as code (TMaaC) into repository
Integration & Testing	Validate OBC ↔ ground station ↔ payload workflows	Run threat model in CI/CD; check mitigations; fuzz communication protocols; perform fault injection
Deployment	Launch and operational mission	Confirm security assumptions hold (e.g., cryptographic keys not exposed; uplink protected)
Operations & Maintenance	Daily operations and periodic software updates	Monitor threat indicators (telemetry anomalies; uplink spoof attempts); secure OTA updates
Decommissioning	End of mission (e.g., controlled re-entry; software shutdown)	Ensure no uncontrolled access (disable ground link; wipe payload storage)

TAB 2. Threat modeling activities across the space system software lifecycle.

Factor	Signature IDS	Anomaly IDS
Requirement type	Detect specific known attack	Detect unknown/new attacks
Application domain	OBC command validation, software updates, crypto checks	telecommands monitoring, payload data anomaly detection
Resource constraints [37]	Low CPU/memory, fits with embedded RTOS	Needs more memory/processing
False positives [37]	Very low	Higher (depends on training quality)
Adaptability	Static, rule-based	Dynamic, can evolve with new behaviors
Verification	Easy to trace to requirement	Harder to prove correctness (probabilistic)

TAB 3. A comparison between signature IDS and anomaly IDS from a space software perspective.

precomputing recovery plans, compromised tasks can be isolated, and critical operations can continue under a safe, low-cost scheduling framework [20]. Research has also explored distributed, security-aware task migration mechanisms, in which computational tasks move dynamically to more secure nodes depending on security requirements [21]. Such mechanisms enable autonomous containment while preserving mission continuity.

Responses can be categorized across multiple levels.

- Automated local responses occur onboard, such as discarding packets that fail validation, switching to minimal operational modes when anomalies are detected, rolling back firmware updates that fail integrity checks, and containing privilege escalation through task isolation.
- Ground-assisted responses provide additional oversight, including command quarantine, operator-in-the-loop decision-making, and forensic analysis of security telemetry. Mission-aware adaptive responses integrate information from both onboard and ground systems to enable graceful degradation, cross-layer defense, and dynamic reconfiguration of software and operational parameters.

Each response is traceable to specific security requirements derived from threat modeling. For instance, detecting a spoofed telecommand triggers a function to identify invalid commands; the corresponding response may involve discarding the command and logging the anomaly. Similarly, a buffer overflow detected by runtime monitors leads to process termination and safe-mode escalation. Malicious or unsigned software updates trigger rollback procedures to verified images, ensuring system integrity and trust.

Implementing effective response strategies poses several challenges. Limited onboard resources restrict the complexity of automated responses, intermittent communications limit ground intervention, and excessive

false positives may disrupt operations more than the attack itself. Consequently, automated responses must be lightweight, deterministic, and rigorously tested to ensure predictable outcomes.

A representative scenario illustrates these principles: a buffer overflow in the telecommand handler is detected via stack canary violations. The response sequence involves terminating the compromised process, restarting the handler in a clean state, logging the event in security telemetry, and, if repeated incidents occur, dropping the OBC into safe mode until ground intervention is possible. Such structured, requirement-driven responses ensure mission continuity while preserving trust in onboard systems. The response can be measured through metrics such as Mean Time to Respond ($MTTR_{es}$), which represents the time from detection to containment action.

4.3. Recovery

Recovery focuses on restoring normal operations and mission capability after a cyberattack has been contained. Unlike the immediate and tactical nature of response, recovery involves mid- to long-term strategic actions that ensure the system resumes full operational functionality while maintaining trust in its integrity. Recovery measures often combine automated onboard procedures with ground-assisted interventions, reflecting the need for both resilience and oversight.

A fundamental component of recovery is the ability to restore software and system configurations to verified, clean states. For example, failed firmware updates can be rolled back to a secure backup image [41], and critical flight configurations can be reloaded from redundant copies stored onboard. In addition, redundant subsystems, such as backup OBCs or sensor units, provide additional layers of fault tolerance, enabling the spacecraft to continue operating despite localized failures [42]. Forensic analysis on the ground plays a crucial role in identifying vulnerabilities and implementing patches that prevent recurrence.

Recovery measures must be embedded throughout the software lifecycle. During the requirements phase, resilience objectives should be defined, such as specifying that "the OBC shall support rollback to the last verified software image." Design and development phases incorporate redundancy into both software and hardware, for instance by maintaining dual firmware images or redundant communication channels. Integration and verification steps test these recovery mechanisms, confirming that the system can reliably revert to clean states after induced faults or simulated attacks.

In-orbit operations involve ground-assisted recovery [43], where the ground station may upload patches, initiate configuration restoration, or re-establish critical services following the containment of a cyber incident. Over time, maintenance and evolution phases integrate lessons learned from prior incidents into upgrades, enhancing both the robustness and the resilience of future missions. By systematically restoring functionality and trust, recovery closes the loop of operational resilience, preparing the system for subsequent detect–respond–recover cycles.

Effectiveness of recovery strategies can be measured through metrics such as mean time to recover ($MTTR_{ec}$), representing the duration from containment to full mission restoration, and mission downtime, denoting the proportion of time the system is degraded due to cyber events. For instance, a well-designed recovery process may achieve rollback to a verified OBC image within a single ground pass, minimizing mission disruption and maintaining operational continuity. By embedding recovery measures across lifecycle phases and combining automated onboard procedures with ground-based oversight, space systems can achieve robust, reliable, and resilient operations in the presence of evolving cyber threats.

4.4. Key takeaways

Detection, response, and recovery are tightly coupled within the context of cyber resilience. An improvement in detection capabilities contributes little if the response mechanism cannot act promptly. For instance, if a spacecraft's IDS flags anomalies within one second, but the corresponding response is manual and requires several hours, the overall resilience gain is negligible. Conversely, response mechanisms without reliable detection are effectively blind and unable to prioritize resources or mitigate threats. Fast recovery plays a complementary role by restoring system functionality. Therefore, while enhancing detection is necessary, it is insufficient on its own; true resilience emerges only when detection, response, and recovery are co-designed as an integrated pipeline with balanced investment across all stages.

FIG 4 illustrates the interaction between detection, response, and recovery in a simplified Markov-chain model:

- $S0 \rightarrow S1$: the system comes under attack with probability p_a.
- $S1 \rightarrow S2$: the attack is detected with probability p_d; if not, the system remains in $S1$, a critical state that may lead to total satellite loss.
- $S2 \rightarrow S3$: once detected, the system initiates a response.
- $S3 \rightarrow S4$: recovery succeeds with probability p_r; otherwise, the system remains in the response phase. $S3$ corresponds to a degraded quality of service (QoS) for the satellite.
- $S4 \rightarrow S0$: after successful recovery, the system returns to normal operation if no permanent damage has occurred.

It is important to note that prevention plays a crucial upstream role by minimizing p_a, thereby reducing the likelihood of entering the critical state $S1$.

5. EVOLVE

Evolve refers to the continuous improvement of cyber resilience by learning from attacks, anomalies, and

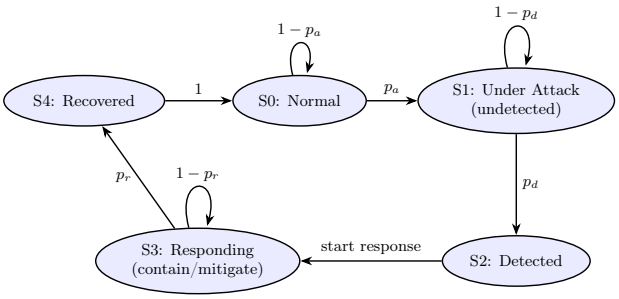

FIG 4. The impact of detection, response, recovery on the system states.

environmental changes, then adapting both system defenses and operational practices. Unlike recovery, which restores nominal operations, evolve aims to strengthen the system against future threats. This process spans the entire software lifecycle as well as in-orbit operations.

A primary element of evolve is threat intelligence integration, where new attack patterns observed during the mission—such as spoofing attempts or novel SDR jamming—are incorporated into detection systems. Signature-based IDS can be updated with new rules, while anomaly-based IDS benefit from retraining with updated telemetry baselines.

Equally critical is adaptive patching and updates. Secure update channels must support firmware upgrades, IDS rule updates, anomaly detection model refinements, and operational policy adjustments. For example, a new buffer overflow exploit detected by ground control can trigger the upload of a patched OBC image to mitigate the threat [2].

An often-overlooked component is incident reporting. Following containment and recovery, anomalies and intrusions should be formally documented, categorized, and analyzed. Structured incident reports provide traceability, inform updates to defenses, and refine operational playbooks. They also align with emerging cybersecurity standards (e.g., ISO/TS 20517:2024 [7], NASA BPG [12]) and support sector-wide resilience through information sharing.

Feedback into requirements and design ensures that lessons learned directly inform future lifecycle iterations. For instance, repeated replay attacks may introduce a new requirement enforcing sequence numbers on all telecommands. Continuous learning extends this process via lightweight onboard adaptation—such as adjusting anomaly thresholds as sensors degrade—and evolving ground-based operational procedures.

Finally, evolve encompasses supply chain considerations. Lessons from incidents, vulnerabilities, or problematic third-party libraries feed into supplier evaluation and procurement practices. Requirements may be updated to mandate vendor assurance, cryptographic provenance verification, or continuous SBOM component scanning. By embedding these measures, space missions enhance both immediate security and long-term adaptability, strengthening defenses for current and future systems.

6. CONCLUSION

Securing space systems requires a holistic, lifecycle-driven approach that integrates cybersecurity into every phase of software and system engineering. Preventive measures, including secure-by-design principles, threat modeling, and rigorous testing, form the foundation for resilient operations. Detection through intrusion detection systems provides real-time awareness of potential attacks, while response mechanisms ensure containment and mitigation without compromising mission-critical functions. Recovery restores system functionality and trust, leveraging redundancy, rollback procedures, and ground-assisted interventions. The evolve phase completes the cycle by incorporating lessons learned, updating defenses, and adapting system processes to emerging threats.

By embedding these principles across the lifecycle, space systems can operate reliably in increasingly complex and adversarial environments. Continuous adaptation and learning transform individual incidents into opportunities to strengthen future missions, ultimately enhancing the overall resilience of space operations.

References

[1] M. Manulis, C. P. Bridges, R. Harrison, V. Sekar, and A. Davis. Cyber security in new space. International Journal of Information Security, 30:287–311. DOI: .org/10.1007/s10207-020-00503-w).

[2] Daniel Lüdtke, Jan Sommer, Carlos Gonzalez, Hendrik Otte, Andreas Lund, Zain H. Hammadeh, Carlo Brokering, and Arnau Prat. Stellar apps: On-board application framework for space missions. 18th Annual Flight Software Workshop. online: https://elib.dlr.de/214032/, 2025.

[3] Nicolò Boschetti, Nathaniel Gordon, and Gregory Falco. Space cybersecurity lessons learned from the viasat cyberattack. In AIAA Ascend, 2022. DOI: 10.2514/6.2022-4380.

[4] Johannes Willbold, Moritz Schloegel, Manuel Vögele, Maximilian Gerhardt, Thorsten Holz, and Ali Abbasi. Space odyssey: An experimental software security analysis of satellites. In 2023 IEEE Symposium on Security and Privacy (SP), pages 1–19, 2023. DOI: 10.1109/SP46215.2023.10351029.

[5] Nikita Yadav, Franziska Vollmer, Ahmad-Reza Sadeghi, Georgios Smaragdakis, and Alexios Voulimeneas. Orbital shield: Rethinking satellite security in the commercial off-the-shelf era. In 2024 Security for Space Systems (3S), 2024. DOI: 10.23919/3S60530.2024.10592292.

[6] Zain A. H. Hammadeh, Mohammad Hamad, Andrzej Olchawa, Milenko Starcik, Ricardo Fradique, Stefan Langhammer, Manuel Hoffmann, Florian Göhler, Daniel Lüdtke, Michael

Felderer, and Sebastian Steinhorst. Designing secure space systems. In 2025 Design, Automation & Test in Europe Conference (DATE), 2025. DOI: 10.23919/DATE64628.2025.10992767.

[7] ISO. ISO/TS 20517:2024 space systems — cybersecurity management requirements and recommendations, 2024. Edition 1. Online: https://www.iso.org/standard/86305.html.

[8] Consultative Committee for Space Data Systems (CCSDS) . Security threats against space missions, 2022. INFORMATIONAL REPORT. Online: https://public.ccsds.org/Pubs/350x1g3.pdf.

[9] IEEE Standard Association. IEEE P3349 standard for space system cybersecurity, 2023. Online: https://standards.ieee.org/ieee/3349/11182/.

[10] IEEE Standard Association. IEEE P3336 standard for space system cybersecurity design, 2023. Online: https://standards.ieee.org/ieee/3536/11916/.

[11] The European Cooperation for Space Standardization (ECSS). ECSS-E-ST-80C – space engineering – security in space systems lifecycles, 2024. Online: https://ecss.nl/standard/ecss-e-st-80c-space-engineering-security-in-space-systems-lifecycles/.

[12] National Aeronautics and Space Administration (NASA) . NASA space security: Best practices guide (bpg), 2024. NASA Software Engineering Handbook.

[13] German Federal Office for Information Security (BSI). Technical guideline BSI TR-03184 information security for space systems - part 1: Space segment, 2023. Online: https://www.bsi.bund.de/.

[14] German Federal Office for Information Security (BSI). Technical guideline BSI TR-031842 information security for space systems - part 2: Ground segment, 2025. Online: https://www.bsi.bund.de/.

[15] Christian R. Prause, Ralf Gerlich, and Rainer Gerlich. Fatal software failures in spaceflight. Encyclopedia, 4(2):936–965, 2024. DOI: 10.3390/encyclopedia4020061.

[16] Daniel Lüdtke, Thomas Firchau, Carlos Gonzalez Cortes, Andreas Lund, Ayush Mani Nepal, Mahmoud M. Elbarrawy, Zain Haj Hammadeh, Jan-Gerd Meß, Patrick Kenny, Fiona Brömer, Michael Mirzaagha, George Saleip, Hannah Kirstein, Christoph Kirchhefer, and Andreas Gerndt. ScOSA on the way to orbit: Reconfigurable high-performance computing for spacecraft. In 2023 IEEE Space Computing Conference (SCC), pages 34–44, 2023. DOI: 10.1109/SCC57168.2023.00015.

[17] Andreas Lund, Zain Alabedin Haj Hammadeh, Patrick Kenny, Vishav Vishav, Andrii Kovalov, Hannes Watolla, Andreas Gerndt, and Daniel Lüdtke. ScOSA system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture. CEAS Space Journal, May 2021. DOI: 10.1007/s12567-021-00371-7.

[18] Mohammad Hamad, Andreas Finkenzeller, Michael Kühr, Andrew Roberts, Olaf Maennel, Vassilis Prevelakis, and Sebastian Steinhorst. REACT: Autonomous intrusion response system for intelligent vehicles. Comput. Secur., 145(C), November 2024. DOI: 10.1016/j.cose.2024.104008.

[19] Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis, and Rolf Ernst. Prediction of abnormal temporal behavior in real-time systems. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, page 359–367, New York, NY, USA, 2018. Association for Computing Machinery. DOI: 10.1145/3167132.3167172.

[20] Zain A. H. Hammadeh, Monowar Hasan, and Mohammad Hamad. RESCUE: A reconfigurable scheduling framework for securing multi-core real-time systems. ACM Trans. Cyber-Phys. Syst., 9(3), August 2025. DOI: 10.1145/3728364.

[21] Mohammad Hamad, Zain A. H. Hammadeh, Davide Alessi, Monowar Hasan, Mert Pese, Daniel Lüdtke, and Sebastian Steinhorst. Enhancing security through task migration in software-defined vehicles. IEEE Internet of Things, 2025. To Appear. DOI: 10.1109/JIOT.2025.3611875.

[22] Ann Yi Wong, Eyasu Getahun Chekole, Martín Ochoa, and Jianying Zhou. On the security of containers: Threat modeling, attack analysis, and mitigation strategies. Computers & Security, 128:103140, 2023. DOI: 10.1016/j.cose.2023.103140.

[23] Gabriele Marra, Ulysse Planta, Philipp Wüstenberg, and Ali Abbasi. On the feasibility of cubesats application sandboxing for space missions. In Second Workshop on the Security of Space and Satellite Systems (SpaceSec), 2024. DOI: 10.14722/spacesec.2024.23033.

[24] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an os kernel. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP '09, page 207–220, New York, NY, USA, 2009. Association for Computing Machinery. DOI: 10.1145/1629575.1629596.

[25] Kari Bingen, Kaitlyn Johnson, Makena Young, and John Raymond. Space threat assessment 2023, April 2023. Online: https://www.csis.org/analysis/space-threat-assessment-2023.

[26] Gregory Falco, Rajiv Thummala, and Arpit Kubadia. WannaFly: An approach to satellite ransomware. In 2023 IEEE 9th International Conference on Space Mission Challenges for Information Technology (SMC-IT), pages 84–93, 2023. DOI: 10.1109/SMC-IT56444.2023.00018.

[27] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. STRIDE-based threat modeling for cyber-physical systems. In 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), pages 1–6, 2017. DOI: 10.1109/ISGTEurope.2017.8260283.

[28] OWASP Top Ten. https://owasp.org/www-project-top-ten/. Accessed: 2025-09-08.

[29] Anna Georgiadou, Spiros Mouzakitis, and Dimitris Askounis. Assessing MITRE ATT&CK risk using a cyber-security culture framework. Sensors, 21(9), 2021. DOI: 10.3390/s21093267.

[30] Eric O'Donoghue, Ann Marie Reinhold, and Clemente Izurieta. Assessing security risks of software supply chains using software bill of materials. In 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C), pages 134–140, 2024. DOI: 10.1109/SANER-C62648.2024.00023.

[31] Hany Abdelmaksoud, Zain A. H. Hammadeh, Goerschwin Fey, and Daniel Lüdtke. DEL: Dynamic symbolic execution-based lifter for enhanced low-level intermediate representation. In 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1–2, 2023. DOI: 10.23919/DATE56975.2023.10137253.

[32] Raffaela Groner, Thomas Witte, Alexander Raschke, Sophie Hirn, Irdin Pekaric, Markus Frick, Matthias Tichy, and Michael Felderer. Model-based generation of attack-fault trees. In Jérémie Guiochet, Stefano Tonetta, and Friedemann Bitsch, editors, Computer Safety, Reliability, and Security, pages 107–120, Cham, 2023. Springer Nature Switzerland. DOI: 10.1007/978-3-031-40923-3_9.

[33] Michael Felderer, Matthias Büchler, Martin Johns, Achim D. Brucker, Ruth Breu, and Alexander Pretschner. Chapter one - security testing: A survey. In Atif Memon, editor, Advances in Computers, volume 101 of Advances in Computers, pages 1–51. Elsevier, 2016. DOI: 10.1016/bs.adcom.2015.11.003.

[34] Erika Pärn, Nikdokht Ghadiminia, Borja García de Soto, and Kwadwo Oti-Sarpong. A perfect storm: Digital twins, cybersecurity, and general contracting firms. Developments in the Built Environment, 18:100466, 2024. DOI: https://doi.org/10.1016/j.dibe.2024.100466.

[35] Yury A. Kuleshov, Kabir Nagpal, Korel Ucpinar, Alisha Gadaginmath, Sanjana Gadaginmath, Katie O'Daniel, Dalbert Sun, Lucas Tan, Nathan Veatch, and Hridhay Monangi. Cyber Attacks on Avionics Networks in Digital Twin Environment: Detection and Defense. DOI: 10.2514/6.2024-0277.

[36] Maonan Wang, Kangfeng Zheng, Yanqing Yang, and Xiujuan Wang. An explainable machine learning framework for intrusion detection systems. IEEE Access, 8:73127–73141, 2020. DOI: 10.1109/ACCESS.2020.2988359.

[37] Vasudev Karthik Ravindran, Sharad Shyam Ojha, and Arvind Kamboj. A comparative analysis of signature-based and anomaly-based intrusion detection systems. International Journal of Latest Technology in Engineering Management & Applied Science, 14(5):209–214, Jun. 2025. DOI: 10.51583/IJLTEMAS.2025.140500026.

[38] Ly Vessels, Kenneth Heffner, and Daniel Johnson. Cybersecurity risk assessment for space systems. In 2019 IEEE Space Computing Conference (SCC), pages 11–19. IEEE. DOI: 10.1109/spacecomp.2019.00006.

[39] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, 1975. DOI: 10.1109/PROC.1975.9939.

[40] Samuel Jero, Juliana Furgala, Max A Heller, Benjamin Nahill, Samuel Mergendahl, and Richard Skowyra. Securing the satellite software stack. In Second Workshop on the Security of Space and Satellite Systems (SpaceSec), 2024. DOI: 10.14722/spacesec.2024.23057.

[41] Nicole Webb, Mark Johnson, and Patrick Saenz. Enhancing satellite cybersecurity through FPGA-based secure boot. In 2025 IEEE Aerospace Conference, pages 1–7, 2025. DOI: 10.1109/AERO63441.2025.11068623.

[42] J. Bouwmeester, A. Menicucci, and E.K.A. Gill. Improving cubesat reliability: Subsystem redundancy or improved testing? Reliability Engineering & System Safety, 220:108288, 2022. DOI: 10.1016/j.ress.2021.108288.

[43] Massimo Tipaldi and Bernhard Bruenjes. Survey on fault detection, isolation, and recovery strategies in the space domain. Journal of Aerospace Information Systems, 12(2):235–256, 2015. DOI: 10.2514/1.I010307.