# Orbital Shield: Rethinking Satellite Security in the Commercial Off-the-Shelf Era

Nikita Yadav*, Franziska Vollmer†, Ahmad-Reza Sadeghi†, Georgios Smaragdakis‡, and Alexios Voulimeneas‡
*Indian Institute of Science, Bangalore, India
Email: nikitayadav@iisc.ac.in
†Technische Universität Darmstadt, Germany
Email: {franziska.vollmer@stud, ahmad.sadeghi@trust}.tu-darmstadt.de
‡Delft University of Technology, The Netherlands
Email: {g.smaragdakis,a.voulimeneas}@tudelft.nl

***Abstract*—Satellites have become part of critical infrastructure utilized for diverse applications, from Earth observation to communication and military missions. Several trends have reshaped satellite deployment and utilization in recent years, making satellite systems more accessible and vulnerable to cybersecurity threats. A notable trend is adopting Commercially Off-the-Shelf (COTS) hardware and software for satellite systems. However, this approach renders satellites susceptible to well-known cyberattacks. This paper presents a comprehensive exploration of attacks on satellite systems, with a specific emphasis on the security aspects of the satellite platform, encompassing both the bus and payload subsystems. The discussion includes existing security defenses that can enhance the security of the satellite platform. Ultimately, we present a real-world security framework designed to improve the overall security of the satellite platform.**

***Index Terms*—Satellite Security, Space Systems Security**

## I. Introduction

In recent years, there has been a growing reliance on satellite systems. Satellites are used in various critical domains, such as telecommunications, navigation, and military operations. Due to their versatility, satellite systems have become an essential component of global networking and information transmission. Satellites are anticipated to play an increasingly vital role in the evolution of telecommunications technology, particularly with the advent of 5G and the forthcoming 6G systems. Therefore, it is crucial to safeguard these systems as they are relied upon. Security risks in space are not limited to physical threats but also extend to malicious manipulation of complex software and communication systems [1].

**Trends.** Several trends have disrupted the space market. Firstly, the rise of small satellites is projected, with approximately 15,000 expected launches between 2021 and 2030, 90% of them being small satellites [2], which are defined by their small size, weight, and cost-effectiveness. Second, to build low-cost satellites, Commercially Off-the-Shelf (COTS) hardware and software are increasingly used [3]. Third, unlike traditional satellites dedicated to a single mission or user, new satellites operate on a multi-tenant model [4]. This facilitates shared access, enabling various entities to leverage the satellite's capabilities for different purposes. However, it makes the satellite system more susceptible to security threats. Lastly, Reprogrammable satellites, exemplified by the standardized OneSat satellites [5], [6], are software defined and fully reprogrammable in orbit. However, this increased flexibility also introduces potential security vulnerabilities.

**Adversary Model.** Satellites often belong to stakeholders ranging from government agencies and defense organizations to private companies and research institutions. This diversity in ownership and operation significantly increases the attack surface of satellite systems. The multitude of stakeholders introduces various points of vulnerability that malicious actors may exploit to compromise satellites. Furthermore, the interconnected nature of satellite systems means that an attack on one satellite can potentially impact others within the same network or constellation. This interconnectedness amplifies the consequences of successful attacks and underscores the importance of robust security measures and collaborative efforts among stakeholders to mitigate risks and safeguard satellite assets and operations.

**Attacks on Satellites.** Cyber threats on satellites include unauthorized access attempts, malware infections, and vulnerabilities in satellite communication protocols and ground-based infrastructure. Satellites have been subject to real-world attacks multiple times [1]. A prominent attack instance occurred during the Ukraine-Russia war, known as the *Viasat attack*. The attackers exploited a security vulnerability in the ground segment, executing a privilege escalation. The malware used in the attack made it impossible for thousands of users to connect to the network [7]. Although real-world attacks on satellites have been infrequent, the increasing reliance on satellite technology and the growing usage of commodity hardware and open satellite architecture and protocols emphasize the importance of proactive cybersecurity measures to safeguard satellite systems against potential attacks and mitigate the risks associated with cyber vulnerabilities.

Satellite attacks can be categorized into two main types: (a) network attacks, focusing on communication links within a satellite infrastructure, including jamming and spoofing, and (b) platform attacks, directed at a satellite's software and hardware. A network attacker can intercept unencrypted or weakly encrypted RF signals, disrupt signals by transmitting on the same frequency with higher power [8], and send deceptive signals to manipulate receiver systems [9]. More sophisticated attacks include satellite platform attacks, which corrupt data, obtain access credentials, and even send malicious updates to the firmware by exploiting software and hardware vulnerabilities [10], [11]. Ensuring the security of the satellite platform is vital, as otherwise, attackers can take control of the satellite. This could have severe consequences, especially in critical domains. As demonstrated in prior works, a satellite can be hijacked to run vulnerable telecommands [12]. An attacker can exploit such vulnerabilities to alter the satellite's orbit, causing it to crash with other satellites. Hence, it poses a significant threat to the space missions. While both network and platform security are crucial, this paper exclusively focuses on platform security due to its relatively lesser emphasis in prior works. For satellite network communication security, other works such as [8], [13], [14] can complement our findings.

**Our Focus: Satellite Platform Security.** Low-Orbit Satellites (LEO) or Cube Satellites (CubeSats) are growing rapidly in the recent years [15], [16]. The CubeSats are generally built using Commercially Off-the-Shelf (COTS) hardware components for cost-savings [17], [18]. Other reasons for using COTS hardware components include availability, rapid technological advancements and reduced satellite manufacturing time. The global satellite COTS components market is estimated to double in the next decade, at a growth rate of 1.21% during the forecast period 2022-2032 [19], [20]. According to reports by Microwave Journal, NASA agrees that using COTS hardware can benefit many satellite applications and guides to test the reliability of COTS hardware for space applications [21]. However, other types of satellites such as Geostationary Orbit (GEO) or Medium Earth Orbit (MEO) may have different design requirements, longer operational lifespans, and higher reliability standards, which could necessitate the use of more specialized or radiation-hardened components.

In the low-cost satellites, COTS hardware and software is employed [3], [22], [23], as exemplified by the use of an ARM-M processor for the bus system and an ARMv9 processor for the payload system in the case of FlatSat [24], and ARM cortex-M3 On-board Controller (OBC) in ESTCube-1 [25]. This leaves the space systems vulnerable to commodity cyber threats, such as crypto-ransomware, malware, and exploiting common software and hardware vulnerabilities. Consequently, the ongoing research in security architectures for cyber-physical systems (CPS) is directly relevant to satellite systems.

We present the requirements for a secure satellite architecture. As modern satellites increasingly utilize COTS hardware and their architecture becomes widely known, we draw parallels between the security considerations of satellite systems and cyber-physical systems (CPS). We explore existing security tools and techniques applied to CPS. We assess the applicability of these methodologies to meet the security requirements in the context of satellite platforms. We conclude our work with a secure satellite design that meets all the specified security criteria.

This white paper focuses on the conceptual and theoretical aspects of the security of satellite platforms. Consequently, it does not include an evaluation or implementation of the proposed security architecture.

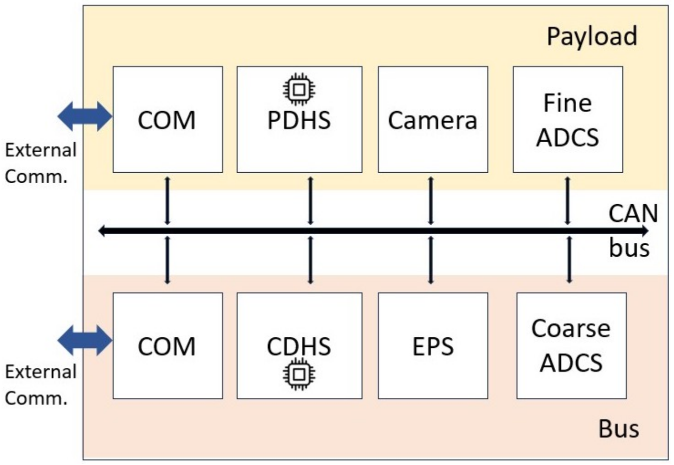

Fig. 1: Satellite architecture illustrating the bus-payload separation.

**Outline.** The paper is organized as follows: Section II overviews the satellite system architecture. Potential attacks on the satellite platform are discussed in Section III. Through analyzing potential attacks, we derive requirements to ensure comprehensive security for a satellite system (Section V-A). We assess existing security tools and techniques in Section V-B to determine their alignment with the defined requirements, incorporating insights from prior satellite research for real-world applicability. Finally, Section V-C introduces a security framework for the satellite platform.

## II. Satellite system basics

Modern satellites feature a bus-payload separation, dividing the satellite into two components: (a) the satellite bus and (b) the satellite payload [26]. The Controller Area Network (CAN) protocol establishes communication between the payload side and the satellite bus through a CAN bus in CubeSat architectures. Consequently, our discussion and analysis will be centred around CubeSats and their CAN bus systems. Figure 1 shows a high-level architecture of the satellite, highlighting the typical components. This paper collectively refers to the satellite bus, payload, and CAN bus as the satellite platform.

### *A. Bus System*

The satellite bus controls the satellite and provides support to the payload side. It includes all the necessary components for the satellite's operation and functionality. It is important to note that the bus system is not reliant on the payload side, although the payload side is dependent on the bus system.

The Command and Data Handling System (CDHS) is the most crucial component of the bus system for ensuring effective data processing and control. The CDHS uses an Onboard Controller (OBC) with a microcontroller and memory. The software running on the OBC is called onboard software (OBSW). The Electrical Power System (EPS) generates, stores, and distributes electrical energy. Meanwhile, the Attitude Determination and Control System (ADCS) stabilizes the satellite's orientation and position in space. The Communication Subsystem (COM) also enables bidirectional communication between the satellite and ground stations.

## *B. Payload System*

The mission's success heavily relies on the satellite's payload. It includes mission-critical equipment, such as cameras, optical receivers, or communication radios, and is accountable for both payload control and mission data processing. The Payload Data Handling System (PDHS) is the central component within the payload, handling communication, receiving control data, and processing the sensor data. The payload side of the satellite offers multi-tenant functionality, i.e., Satellite-as-a-service (SaaS) [4]. Multiple users can access and execute their code on the satellite. The satellite resources, such as peripherals, are shared by all the users.

## *C. Shared Communication Channel*

The Controller Area Network (CAN) bus protocol is commonly used to facilitate communication between various devices and subsystems within a Cubesat satellite. The CAN protocol utilizes a two-wire interface mode for communication on a shared bus. The two wires are known as 'CAN Low' and 'CAN High'. Electronic Control Units (ECUs) are connected to the differential data lines and communicate with each other via the differential bus. The CAN two-wire broadcast topology enables cost-effective connections of multiple nodes to a single bus, reducing system and cable costs. The destination address is transmitted via the bus. ESA states that using a CAN bus can reduce a satellite's weight and mass by up to 18% [27]. This is especially crucial for small satellites, where board space and size are significant factors. The technical advantages of a CAN bus make it an ideal choice for satellite use and its ability to protect systems from high-voltage transients and avoid ground loops through an isolated CAN bus [28].

## III. Attacks on satellite platform

Attacks on satellite platforms encompass various scenarios, including attacks on the onboard controller software in the bus subsystem and software within the payload subsystem and exploit targeting vulnerabilities in the CAN bus protocol. This section outlines these attacks and summarizes relevant prior research. We classify the attacks into two types: (1) targeting the satellite bus and payload and (2) targeting the CAN bus.

Table I provides a comprehensive overview of the various types of attacks, including their specific characteristics, techniques, components affected, vulnerabilities exploited, impact, threat models, and likelihood of successful execution.

## *A. Satellite Bus and Satellite Payload*

*a) Denial of Service (DoS):* Denial-of-service attacks are the most common form of satellite attacks and can affect both the software on the bus and the payload. These attacks can impact the availability of a satellite and block network-based services [9], [11]. A DoS attack can be caused by data overload, which involves sending too much data to the satellite [12]. Some satellite systems employ a Broadcast Controller (BC) as a central hub responsible for dispatching commands to various Remote Terminals (RTs) [13]. A DoS attack aims to prevent the execution of commands by overloading the BC's command code. As a result, the RTs intended to receive these commands become unable to execute them due to the excessive load imposed on the BC.

*b) Data Corruption:* In this attack, the attacker intentionally alters or destroys data in a computer system or network. The attacker may manipulate satellite data to access the bus software. For instance, the attacker can access payload or flight-critical data and access data of the bus. The data falsified by the attacker can spread throughout the bus system and jeopardize the security of the satellite platform [9], [12], [13].

*c) Backdoor Attacks:* Backdoor attacks aim to exploit existing backdoors in a satellite's software or hardware or to create new ones. This can be achieved by injecting code into the software. Using these backdoors, attackers can gain unauthorized access to a satellite and potentially execute further attacks [9].

*d) Cryptographic Key Replacements:* Attackers can gain control of the satellite and its communication channel by changing the cryptographic keys with malicious ones and exploiting vulnerabilities in the key management system, resulting in loss of control by the operator [9].

*e) Memory Corruption:* In these attacks, the attacker manipulates the system's memory to install malicious code or commands on the satellite. The injected commands are then executed during events such as a system restart to ensure the code remains on the system even after a reset [9]. Memory corruption is also possible in TC. Memory vulnerabilities such as buffer overflows can be exploited by manipulating the memory [12]. For instance, in the file rename function, the new filename is copied into a buffer with a static size without performing any size verification. Similarly, in the AVR32-QEMU the UHF-TC-Fetcher interface uses a strcat function to copy a character string from a telecommand (TC) to a buffer. The telecommand (TC) can be larger than the static buffer, thus causing a buffer overflow. Such attacks allow the attacker to execute arbitrary code and gain control of the satellite. Another method of achieving memory corruption is through memory leakage attacks in satellite security [32], [33], [34]. These occur when programmes or systems fail to properly release allocated memory. Over time, this can result in memory utilisation, performance degradation and potential security risks. In satellite systems, memory leakage vulnerabilities can jeopardise critical functions and data. There are different types of memory leakage attacks, including buffer overreads. In this type of attack, a program reads beyond the boundaries of an allocated buffer, which can expose sensitive information stored in neighbouring memory areas. Another form of attack is the exploitation of memory leaks, which occur when programs reserve memory but do not subsequently release it. This can

TABLE I: Overview of security threats and attack techniques in satellites.

| **Attack Type** | **Attack Technique** | **Components Attacked** | **Used Vulnerabilities** | **Impact** | **Threat Model** | **Attack Exploitability** |
|---|---|---|---|---|---|---|
| Denial of Service [9], [11] [12] [13], [29] | Jamming, Internet Control Message Protocol (ICMP) echo requests (Ping-flooding) | Satellite Bus, Satellite Payload, CAN Bus | Deficiencies in security measures in satellite communications infrastructure of ground stations and remote sites | Internal communications vulnerabilities of Satellite like high bit error rate, high connection delays, control of transmit power, and large round-trip delays. | External attacker with capacity to overwhelm network with flood of requests. | High |
| Data Corruption [30], [9], [12], [13] | Spoofing | Satellite Bus, Satellite Payload | Lack of robust data integrity schemes (e.g., hashing, check values, and digital signatures) and inadequate encryption of time data | Compromise of integrity and confidentiality of data, specifically payloads or flight-critical information, has potential to cause disruption to satellite operations. | Insiders with access to satellite systems or external actors with advanced capabilities. | Medium |
| Backdoor Attacks [30], [9], [31] | Code Injection | Satellite Bus, Satellite Payload | Rare monolithic satellite software structure and inclusion of third-party code for various components | Granting unauthorized access to satellite may potentially result in further attacks and data breaches. | Attacker has system/network access and detailed understanding of architecture and vulnerabilities of system. | High |
| Cryptographic Key Replacement [9] | Over-the-air rekeying procedures | Satellite Communication System | Deficiencies in strategy for key management | Loss of operator control over satellite, compromised communication security, and potential data manipulation. | State-sponsored attackers syndicates with resources to exploit communication protocols | Low |
| Memory Corruption [9], [12], [32], [33], [34] | Buffer-overflow, Code Injection, Exploitation of memory leaks, Misuse of printf-like functions, Buffer-overreads | Satellite Bus | Inadequate security checks or limitations on input data of program or application. Disclosure of sensitive program data (e.g., private keys or code pointer locations) that has been either inadvertently or deliberately disclosed. | Installation of malicious code and subsequent takeover of control of satellite, compromise data integrity and confidentiality, bypassing system's security mechanisms. | Advanced threat actors with capability to exploit system vulnerabilities and execute malicious code | Medium |
| Brute Force Attacks [9], [35] | Brute-Forcing | Satellite Payload | Weak manufacturer security passwords | Unauthorized access to sensitive information, increased risk of unauthorized system control or data manipulation, and ability to update configuration files and control UHF communications | External attacker with no specific knowledge of this particular system. | Low |
| Bypass Access Control [12] | Execution of arbitrary telecommands (TCs) | Onboard Computer (OBC) | Deactivated encryption and authentication functions | Complete compromise of satellite control and potential disruption of mission objectives or functions. | External actors with knowledge of satellite control systems. | Medium |
| Malicious Firmware Updates [12] | Transmission of critical TCs to the CDHS, modification of existing or new files | Satellite Bus | Outdated or insecure firmware | Installation of unauthorized software, potential takeover of satellite control, disruption of normal operations. | Malicious insiders or external actors with access to firmware update mechanisms. | High |
| Unauthorized Node Injection [36], [37] | Man-in-the-Middle Attacks (MITM), Chosen-Plaintext-/Known-Plaintext-Attacks | CAN Bus | Absence of effective authentication mechanisms in CAN bus | Potential manipulation of critical communication systems, compromise of satellite functions, Data susceptible to interception by unauthorized nodes. | Attackers with access to CAN bus. | High |
| CAN-Bus-Message Tampering [30] [36], [38] | Impersonation Attacks, Replay Attacks, Sniffing Attacks | CAN Bus | Absence of effective authentication mechanisms in CAN bus | Manipulation of Satellite data and instrument cluster data | Attackers with ability to forge messages | Low |

result in a shortage of resources. In addition, misuse of printf-like functions or logging mechanisms can inadvertently expose sensitive data in logs or output streams. The impact of such attacks on satellites is significant. Leaked data can jeopardise the confidentiality of satellite communications, payload information and system configuration. Unreleased memory can lead to resource scarcity and affect the overall stability and performance of the system. Furthermore, memory leaks could potentially reveal the critical state of the system, thereby enabling attackers to circumvent security mechanisms.

*f) Brute Force Attacks:* Attackers may attempt to gain access to a satellite's platform data, including login information for user accounts or cryptographic keys. This could allow them to access the satellite's software and potentially carry out further attacks. Brute force attacks are commonly used to obtain this information. The payload controller is vulnerable due to the manufacturer's weak security passwords. If attackers gain access to these passwords, they can update configuration files and control UHF communications [9], [35].

*g) Bypassing Access Control:* Prior works show that satellites lack proper access control mechanisms [12]. Specifically, the COM radio is directly connected to the OBC in the satellite's architecture, using the Cubesat Space Protocol (CSP), which has deactivated encryption and authentication functions. External attackers can control the satellite through the execution of arbitrary telecommands (TCs).

*h) Malicious Firmware Updates:* Another threat is uploading malicious firmware images to the satellite, e.g., via the flash file system in OPS-SAT [12]. This can be achieved by modifying existing files or creating new ones. Alternatively, an attacker controlling the PDHS can send critical TCs to the CDHS to initiate a malicious firmware update.

### B. CAN Bus

The satellite's internal connections are formed using the CAN bus, making it a pivotal security aspect. Current literature on satellite attacks often neglects to address vulnerabilities in the CAN bus. Therefore, we incorporate insights from attacks on the CAN bus in other systems, such as automotive systems, to enhance our understanding of potential threats.

*a) Unauthorized Node Injection :* The Controller Area Network (CAN) bus has some design limitations that represent weak points. One of these limitations is the lack of authentication, which allows any node to join the network and participate in communication. As the data in the CAN bus is transmitted unencrypted, an unauthorized node can intercept and understand the data. Furthermore, the unauthorized node can disseminate incorrect data in the network [36], [37].

*b) Denial of Service:* Denial of Service attacks are a possible threat to the CAN bus. Three basic attack methods can be distinguished. The first method involves sending an excessive number of request messages to a parameter group number (PGN) to cause an overload. The second method is the sending of a manipulated false request to send (RTS) to generate a buffer overflow at the receiver. The final process of a DoS attack discussed in prior works involves keeping connections open by sending Clear-to-Send (CTS) messages and occupying the entire network [39].

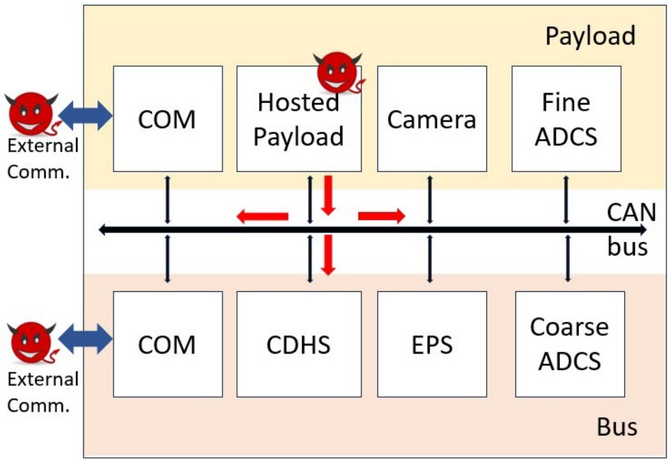

Fig. 2: Entry points for an adversary. An adversary can hijack satellite's external communication to send arbitrary commands and upload a malware on the payload using multi-tenant functionality.

*c) CAN-Bus-Message Tampering:* Attackers can forge or change messages exchanged on the CAN bus. Including impersonation attacks, where the attacker forges or modifies messages after they have been confirmed as valid by the recipient. It also includes replay attacks, in which benign messages are used for fraudulent purposes, and sniffing attacks, in which the attacker intercepts the content of CAN bus messages [36], [38].

## IV. Threat Model

All satellite platform components are susceptible to attacks, with three entry points for attackers, as shown in Figure 2: (1) sending malicious commands via hijacked external communication channels, (2) uploading malicious code on the satellite payload, and (3) exploiting hardware backdoors.

We assume that the attacker possesses comprehensive knowledge of the satellite, including its architecture, communication protocols, hardware, and software components. The attacker can be located on the network and send vulnerable telecommands. Additionally, the attacker can exploit the satellite-as-a-service feature to upload their malicious code onto the satellite payload. We assume that all other software on the satellite are benign but vulnerable. The attacker can send vulnerable commands to exploit the vulnerabilities in the other satellite components. However, we do not consider hardware backdoor attacks in this work.

## V. Securing the satellite system

Modern satellite systems use COTS hardware and software [3]. This introduces several security challenges and threats. Firstly, COTS components are designed for general-purpose applications and may not incorporate the stringent security measures necessary for satellite operations. This lack of built-in security features makes COTS components more susceptible to cyber threats, including malware infections, unauthorized access, and data breaches. Additionally, the widespread availability of COTS hardware and software means that potential attackers have easier access to the same components used in satellite systems. This accessibility allows malicious actors to analyze and exploit vulnerabilities in COTS hardware to compromise satellite operations and data integrity. Further, COTS components are sourced from diverse supply chains with varying levels of security scrutiny. Malicious actors may attempt to infiltrate the supply chain to introduce counterfeit or tampered components, posing significant risks to the security and reliability of satellite systems. The evolving nature of cybersecurity threats means that satellite systems using COTS hardware and software must continuously adapt and respond to emerging security risks. Regular security assessments, updates, and patches are essential to effectively mitigate the evolving threat landscape.

The security of satellite platforms concerns various aspects, such as integrity and isolated software execution, authentication, and access control. The satellite systems are engineered with real-time constraints, employing low-power components with restricted computational and memory capacities. Consequently, security solutions must prioritize efficiency and avoid burdening system resources significantly.

Despite these challenges, there is an opportunity to leverage insights and best practices from the cybersecurity domain, particularly in Cyber-Physical Systems (CPS), to enhance the security measures of satellites. CPS, like satellites, often rely on COTS components and face similar security challenges in ensuring the integrity and resilience of interconnected systems. Like satellite systems, CPS are crafted for specific tasks and tailored for optimal low power consumption. By drawing parallels between satellite systems and CPS, satellite engineers, and security experts can explore and adapt security practices, such as secure execution, intrusion detection systems, secure communication protocols, and access control mechanisms, to mitigate security risks in satellite operations.

This section initially defines the requirements for a secure platform architecture. We then explore established security tools and methodologies designed to address these security challenges and analyze their applicability to satellite systems. The existing solutions and research proposals have their own set of advantages and drawbacks, both in general and specifically regarding deployment in the satellite context. Hence, we integrate the beneficial mechanisms for our secure satellite platform framework.

## A. Security Requirements

We outline the requirements for a security solution for satellite systems.

- R1: Compatibility - The architecture is engineered to be compatible with satellite systems' current design and hardware.
- R2: Secure Operation - The architecture guarantees the confidentiality of critical data and integrity of operations, including command handling and firmware updates.
- R3: Multi-tenant Functionality - The architecture supports the secure simultaneous software execution from multiple users, preventing interference between them.
- R4: Proof of integrity: The architecture supports attestation of the integrity of operations, including command handling, firmware updates, and isolated parallel execution of multiple payload software.
- R5: Secure Communication: The architecture facilitates secure communication on shared CAN bus, incorporating access control measures for resource usage and network monitoring.
- R6: Availability: The system and services are operational and accessible by legitimate users in a timely manner.

## B. Existing Security Tools and Techniques

We discuss the existing security defenses which can meet the security requirements discussed above. Table II provides an overview of these defenses. However, note that our paper focuses exclusively on the security of satellite platform. As availability can be compromised by network attacks such as DoS, we will not be discussing defenses against these attacks in this paper.

**Ensuring Secure Operations: Memory Safety Measures.** Memory vulnerabilities, such as stack buffer overflow, heap overflow, use-after-free, and corruption of code pointers and data variables, can be exploited by malware and other security threats to corrupt program data [71]. The memory vulnerabilities can be exploited to deviate the program control flow to execute arbitrary code sequences, known as code-reuse attacks or control-flow hijacking attacks. These vulnerabilities can be present in both the satellite firmware and the payload software.

We can leverage prior techniques to secure operations. These techniques can be divided into software-based and hardware-based solutions. Examples of software based defenses are stack canaries, DEP, ASLR, CFI and softbound. DEP and ASLR [71] are OS-based memory defenses that ensure code pages are non-writable and data pages are non-executable, while also randomizing the address space layout of the process with each run. Stack canary is a random data inserted in the program's stack at runtime to protect the integrity of the control data such as return addresses. CFI [40], [72]–[77] ensures that the program's execution follows a predefined control flow graph, and deviations from this graph are treated as potentially malicious. It is typically achieved by adding runtime checks and validation to detect any attempt to divert the program's control flow to unintended locations. Softbound [78] ensures memory safety in programs by employing fat pointers. These fat pointers maintain both the base and bound information for each pointer and include code to perform runtime bounds checks, effectively preventing out-of-bounds accesses. Hardware-based memory safety defenses include hardware-assisted CFI [79], ARM

TABLE II: Overview of existing security tools and techniques.

| Defense Type | Approaches | Implementation Efforts | Performance Overhead | Applicability to Satellites |
|---|---|---|---|---|
| Memory Safety | Software-based: Canaries, DEP, ASLR, CFI [40], Softboud; Hardware-based: Intel CET, ARM pointer authentication | Low | Low | Software-based approaches are applicable |
| Memory Isolation | Software-based: Trustvisor [41], ViSA [42], Sanctuary [43]; Hardware-based: SANCUS [44], TrustLite [45], TyTAN [46], OPEC | Medium | Medium | software-based approaches are applicable |
| Attestation | Software-based: SWATT [47], Pioneer [48], VIPER [49]; Hybrid: SMART [50], VRASED [51], RATA [52], IDA [53]; Hardware-based: CFLAT [54], OAT [55], BLAST [56], Lo-FAT [57], ATRIUM [58], Litehax [59] | Low | Medium | All software-based and few hardware-based approaches given secure hardware is present (e.g. CFLAT, OAT, BLAST) are applicable |
| Isolation using Trusted Execution Environment | ARMLock [60], SHREDS [61], ERIM [62], CERBERUS [63], Donky [64], Capacity [65], Hodor [66] | Medium | Medium | Applicable given that secure hardware such as Trustzone is present |
| Authentication and Access Control | VeCur [67], IDS [68], IPS [69], EC-SVC [70] | Low | Low | Applicable |

pointer authentication [80] and memory tagging [81].

**Supporting Multi-tenant Functionality: Memory Isolation Techniques.** Memory isolation ensures that each software instance operates within its designated memory space, preventing unauthorized access or interference from other processes. This helps contain security breaches and limits the impact of potential vulnerabilities in one software instance on others, which is especially relevant in a multi-tenant setting. This can be achieved using either the software or the hardware. Examples of hardware-based memory isolation techniques include SANCUS [44], TrustLite [45], TyTAN [46], and OPEC [82]. SANCUS [44] implements program-counter-based access control and adds new CPU instructions to setup different modules. In contrast to SANCUS, TrustLite [45] effectively manages memory access violations and hardware interrupts without resorting to CPU state and memory resets. TrustLite employs an execution-aware Memory Protection Unit (MPU) in the hardware to enforce isolation. Nevertheless, a drawback is that it mandates all software to be loaded and their isolation policies configured during the boot time. On the other hand, TyTAN extends the idea of TrustLite and offers dynamic task configuration with real-time guarantees, providing a more flexible approach compared to TrustLite's boot-time constraints. OPEC [82] partitions programs into logically independent tasks and enforces privilege and resource isolation at runtime using hardware-supported mechanisms. Although the hardware-based isolation mechanisms offer an efficient solution, they necessitate hardware modifications and lack direct adaptability.

Memory isolation can also be implemented in software. Memory-protected hardware and hardware for virtualization can enable efficient software implementations of the isolated execution environments, making it more suitable for the multi-tenant payload system. Some examples are Trustvisor [41], ViSA [42], and Sanctuary [43]. Trustvisor [41] can be leveraged to implement oracle-like properties, such as cryptographic primitives, in an application. Further, few recent works have employed virtualization for securing satellite systems. ViSA (Virtualised Space Applications) [42] demonstrates a fault tolerance system leveraging the Xen Hypervisor. It highlights that COTS hardware is highly susceptible to radiations, causing single-event effects. The flight software tasks are replicated, and results are compared using the ViSA middleware software to detect faults caused by radiation. Sanctuary [43] proposes a new secure spacecraft architecture leveraging virtualization that executes onboard software and the workloads from different vendors in isolated virtual machines. The hypervisor assigns the hardware to the VMs, including CPU, memory, and peripheral devices. It not only provides isolated execution but also implements remote attestation, a watchdog, and a device manager for access control, thereby meeting all the requirements except backward compatibility (R1). Other works include Lua-based virtualization [83] and a micro hypervisor for flight computer [84].

**Providing Integrity Assurance: Attestation Techniques.** Remote attestation technique allows a remote verifier to establish trust in the integrity of a remote prover. This mechanism can be used to measure the program execution on a remote device, the measurements can then be verified by the verifier. Remote attestation techniques can be classified into three main categories: software-based, hybrid, and hardware-based. Software-based techniques do not depend on hardware security features and instead utilize a custom checksum function implemented entirely in software. These techniques rely on precise timing measurements, which are effective in scenarios with minimal and consistent communication delays between the verifier and the prover, such as between peripherals and a host CPU. Examples of software-based techniques include SWATT [47], Pioneer [48], and VIPER [49]. Hybrid techniques merge software with trusted hardware for integrity measurements. SMART [50], VRASED [51], RATA [52], and IDA [53] are prominent examples of hybrid approaches. SMART [50] employs a measurement routine in ROM with exclusive access to a secret key but protects only a single read-only application and does not handle memory access violations and hardware interrupts. VRASED [51] proposes formally verified remote attestation and stores the attestation code and secret key in ROM while using program counter-based memory access control to protect the key. RATA [52] and IDA [53] prevent time-of-check-time-of-use (TOCTOU) attacks, with IDA also handling interrupts. Other hybrid approaches like CFLAT [54], OAT [55], and BLAST [56], instrument the program binary to record control-flow path measurements and detect control-flow hijacking attacks. CFLAT measures the executed path and stores basic block IDs in secure storage, while BLAST minimizes runtime overhead by using the Ball-Larus Path profiling technique and buffering path measurements in an in-process isolated protected memory region. OAT ensures data integrity by adding value-based define use checks. Hardware-based approaches, such as Lo-FAT [57], ATRIUM [58], and Litehax [59], utilize hardware extensions for efficient program measurement recording.

**Utilizing Trusted Execution Environments (TEEs) for Enhanced Security.** Many processor architectures for MCUs provide secure execution, exemplified by technologies like ARM Trustzone [85]. It offers a trusted execution environment for securing program execution and memory. The hardware-enforced isolated execution environment and secure storage present in these architectures can be used to run security-critical applications in a secure environment and design a comprehensive system-wide security monitor. Another valuable hardware feature available in various architectures to strengthen security is memory protection domains. This feature enables the creation of in-process protected memory regions, safeguarding critical data such as cryptographic keys and shadow stacks. Notable contributions in this domain include ARMlock [60], SHREDS [61], ERIM [62], CERBERUS [63], Donky [64], Capacity [65], and Hodor [66]. These commodity hardware features can be leveraged to enhance the security of the satellite systems.

**Implementing Access Control and Authentication for Secure Communication over CAN Bus.** The Controller Area Network (CAN) operates on a broadcast-based bus system and lacks inherent message authentication mechanisms. This aspect of CAN bus security has been extensively studied, particularly in applications like automotive systems. VeCur [67] introduces a message authentication scheme and a trust group-based structure to enforce access control. It does so by implementing an additional security layer between the original CAN interface and the application, by wrapping the original CAN APIs with additional security features. Lokman et al. [68] implements an Intrusion Detection System (IDS) within the CAN bus network for automotive applications, providing a comprehensive discussion on IDS literature and a taxonomy specific to CAN bus networks. Abott et al. [69] propose a real-time Intrusion Prevention System (IPS) that actively monitors the CAN bus, identifying and eliminating malicious messages. It employs control codes to detect and prevent replay attacks and invalid messages. Addressing confidentiality concerns in the CAN bus network, EC-SVC [70] proposes a fine-grained, attribute-based access control system to safeguard message confidentiality from potential attackers and unauthorized users.

**Discussion.** We note that many of the approaches from existing security defenses are applicable to satellite systems, as illustrated in Table II. Specifically, software-based defenses for memory safety can be easily incorporated into satellite firmware and payloads to prevent memory exploits. These approaches have been successful in preventing attacks and incur low performance overheads. To ensure protection in a multi-tenant setting, satellites can leverage software-based memory isolation techniques without incurring high overhead, as these have been successfully applied not only in CPS but also in satellite settings as well [43]. To verify the integrity of operations, satellite systems can leverage either software-based or hardware-based techniques that do not require changes to the hardware and only require the presence of secure hardware, which is likely to be present in satellite systems. If trusted hardware is available, it can be used to enhance security further and implement in-process memory protection regions in untrusted applications. Finally, the works on CAN bus security are directly applicable to satellite systems since they solve similar problems, and these defenses do not require any special hardware and can be implemented entirely in software. Based on the analysis of the security requirements and applicable defenses, we propose a security architecture for satellites.

### C. Secure Satellite Architecture

We propose a security design that fulfills all the requirements listed in Section V-A and leverages the existing security defenses. To facilitate compatibility with the current satellite architecture, our design does not require any hardware changes. Figure 3 shows the core components of our security architecture. We implement memory safety defenses in the firmware and payload software to prevent memory exploits. We implement a security agent using the Trusted Execution Environment (TEE), and leverage it to enable secure boot, isolation, attestation, and access control over the CAN bus as discussed below.

**Secure Platform Setup.** To build a secure system, we must ensure that all components and their configurations are accurately set during startup. We achieve this through the implementation of secure boot. The initiation of secure boot is entrusted to the code stored in read-only memory (ROM), chosen for its inherent security. The ROM is responsible for establishing the memory controller and invoking *init_task*, which, in its execution, configures the filesystem and mounts devices. *Init_task* verifies the boot configuration file by computing its hash. This sequential process continues, with each component calculating a cryptographic hash of the subsequent component and comparing it against a reference

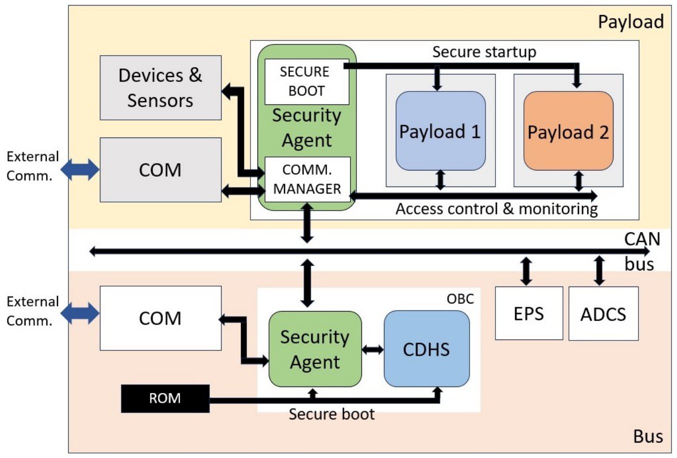

Fig. 3: Secure satellite architecture Overview

value to confirm its unaltered state—the implementation of secure boot guarantees that the system commences with unmodified and trustworthy code. Besides loading validated software, the secure boot process is responsible for setting up essential system configurations, including memory management policies and interrupt vector tables. Additionally, it secures these policies against modification by untrusted software, including the operating system.

**Secure Operations.** We employ memory defenses such as stack canaries, DEP, ASLR and CFI [40], [86] to protect the firmware and payload from runtime memory exploits. We can create secure regions for storing critical data, such as cryptographic keys. This can be accomplished in the software through software-fault isolation [87] or by leveraging hardware features like memory protection domains [60].

**Secure Multi-tenant Functionality.** We assume that the payload hardware is equipped with a trusted execution environment (TEE), such as ARM Trustzone. We use this TEE to secure the operating system code and critical data, such as page tables, from malicious attacks for the complete security of the system (e.g. [88]). Further, all the processes are isolated via memory virtualization, ensuring payloads from different users run as separate isolated processes.

**Secure communication over CAN bus.** In a satellite system, various sensors and devices, such as cameras, sun sensors, reaction wheels, and gyroscopes, are interconnected through the CAN bus. Without proper access control mechanism on the CAN bus, a malicious satellite user could potentially gain unauthorized access to critical peripherals, like reaction wheels, leading to the hijacking of satellite control. Additionally, the CAN bus serves as a broadcast communication channel, allowing data from a sensor to be read by all payloads. Since multiple tenants may exist, data confidentiality becomes a concern, motivating the need for data confidentiality. The security of the CAN bus has been extensively researched in other domains. Our objective is to establish control isolation at the system level across various components. To achieve this, our design incorporates a security agent within the Trusted Execution Environment (TEE) to enforce access control policies. In addition, the payload software is required to specify its peripheral accesses in a manifest file upon startup. Prior to software loading, these specifications undergo verification to ensure correctness. Additionally, each device tags its data with a unique identifier. The security agent includes a communication manager that verifies the origin of data, forwarding it only to software components with the requisite device access permissions, thereby ensuring a secure and controlled data flow. Furthermore, a similar security agent deployed on the satellite bus can oversee communication and implement similar security protocols as shown in Figure 3.

**Attestation-as-a-Service.** By leveraging the security agent, we can implement an attestation service to generate a proof of operation integrity. The security agent can periodically measure the system state such as process memory and system configurations. It can also record the program execution traces at runtime. This information can be leveraged to investigate which software is running and verify the execution and data integrity. The verification process involves analyzing the collected system state data to detect anomalies indicating unauthorized modifications or tampering attempts. By comparing the current system state with the established baseline, security agents can identify discrepancies in software execution, configuration settings, or data integrity.

## VI. Conclusion

The increasing reliance on Commercial Off-The-Shelf (COTS) software and hardware in satellite systems presents a critical challenge as it exposes satellites to a wide range of commodity cyber threats. Our paper highlights this growing vulnerability within satellite platforms, outlining prerequisites for a secure satellite architecture. We explore how existing security tools and techniques from the cyber-physical systems (CPS) domain can be adapted and applied to enhance the security of satellites. We integrate beneficial techniques and put forward a security framework for satellite platforms, intending it to be a foundation for future research and development.

## Acknowledgment

This work has received funding from the European Union's research and innovations program Horizon Europe under grant agreements "MLSysOps" (101092912) and "SEPTON" (101094901), the European Research Council (ERC) under Starting Grant "ResolutioNet" (679158), the Dutch National Growth Fund under the 6G flagship project "Future Network Services", and the Deutsche Forschungsgemeinschaft (DFG) SFB 1119 CROSSING/236615297.

## References

[1] Space and Cyber Secuity, "Space Attacks Open Database Project," https://www.spacesecurity.info/en/space-attacks-open-database/, 2024.

[2] German Aerospace Centre, "Future market small satellites," https://event.dlr.de/en/ila2022/zukunftsmarkt-kleinsatelliten/, 2023.

[3] B. Nussbaum and G. Berg, "Cybersecurity implications of commercial off the shelf (cots) equipment in space infrastructure," *Space infrastructures: From risk to resilience governance*, pp. 91–99, 2020.

[4] Exodus Orbitals. Satellite-as-a-service: a new approach for space industry. Exodus Orbitals. [Online]. Available: https://www.exodusorbitals.com/files/whitepaper.pdf

[5] European Space Agency, "Reprogrammable satellite launched," https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Reprogrammable_satellite_launched, 2021.

[6] ESA, "Reprogrammable satellite design finalised," https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Reprogrammable_satellite_design_finalised, 2021.

[7] N. Boschetti, N. G. Gordon, and G. Falco, "Space cybersecurity lessons learned from the viasat cyberattack," in *ASCEND 2022*, 2022, p. 4380.

[8] M. Manulis, C. P. Bridges, R. Harrison, V. Sekar, and A. Davis, "Cyber security in new space: analysis of threats, key enabling technologies and challenges," *International Journal of Information Security*, vol. 20, pp. 287–311, 2021.

[9] R. Peled, E. Aizikovich, E. Habler, Y. Elovici, and A. Shabtai, "Evaluating the Security of Satellite Systems," 2023.

[10] G. Falco, A. Viswanathan, and A. Santangelo, "CubeSat Security Attack Tree Analysis," in *2021 IEEE 8th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2021, pp. 68–76.

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

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

[13] D. He, X. Li, S. Chan, J. Gao, and M. Guizani, "Security analysis of a space-based wireless network," *IEEE Network*, vol. 33, no. 1, pp. 36–43, 2019.

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

[15] E. Kulu, "NANOSATELLITE & CUBESAT DATABASE," https://www.nanosats.eu/database, 2023.

[16] M. A. Swartwout, "CubeSat Database," https://sites.google.com/a/slu.edu/swartwout/cubesat-database, 2023.

[17] F. Davoli, C. Kourogiorgas, M. Marchese, A. Panagopoulos, and F. Patrone, "Small satellites and cubesats: Survey of structures, architectures, and protocols," *International Journal of Satellite Communications and Networking*, vol. 37, no. 4, pp. 343–359, 2019.

[18] P. Shah and A. Lai, "Cots in space: From novelty to necessity," in *35th Annual Small Satellite Conference*, 2021.

[19] BIS Research, "Satellite commercial-off-the-shelf components enabling innovation in space technology," https://bisresearch.com/news/satellite-commercial-off-the-shelf-components-enabling-innovation-in-space-technology-, 2023.

[20] S. Research, "Satellite commercial-off-the-shelf components market," https://straitsresearch.com/report/satellite-commercial-off-the-shelf-components-market, 2023.

[21] M. Journal, "The great debate: Should cots components be used in space?" https://www.microwavejournal.com/articles/38974-the-great-debate-should-cots-components-be-used-in-space?page=2, 2022.

[22] R. Doyle, R. Some, W. Powell, G. Mounce, M. Goforth, S. Horan, and M. Lowry, "High performance spaceflight computing (hpsc) next-generation space processor (ngsp): a joint investment of nasa and afrl," in *Proceedings of the Workshop on Spacecraft Flight Software*, 2013, pp. 1–19.

[23] K. Karvinen, T. Tikka, and J. Praks, "Using hobby prototyping boards and commercial-off-the-shelf (cots) components for developing low-cost, fast-delivery satellite subsystems," *Journal of Small Satellites*, vol. 4, no. 1, pp. 301–314, 2015.

[24] eoPortal, "Flatsat (ground-based testbed for cubesats)," https://www.eoportal.org/other-space-activities/flatsat#overview, 2021.

[25] ESA eoPortal, "Estcube-1 -2 (estonian student satellite-1 -2)," https://www.eoportal.org/satellite-missions/estcube-1#estcube-1---2-estonian-student-satellite-1---2, 2024.

[26] J. R. Wertz and W. J. Larson, *Space Mission Engineering: The New SMAD*. Microcosm Press, 2011.

[27] R. E. America, "White paper: Using can bus serial communications in space flight applications," Renesas Electronics America, Tech. Rep. [Online]. Available: https://www.renesas.com/us/en/document/whp/using-can-bus-serial-communications-space-flight-applications

[28] S. Singer, "White paper: Isolated can bus for small satellite applications," Intersil Space and High Reliability Products, Renesas Electronics Corp., Tech. Rep., February 2019.

[29] M. Usman, M. Qaraqe, M. R. Asghar, and I. Shafique Ansari, "Mitigating distributed denial of service attacks in satellite networks," *Transactions on emerging telecommunications technologies*, vol. 31, no. 6, p. 9, 2020.

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

[31] L. del Monte, "Towards a cybersecurity policy for a sustainable, secure and safe space environment," in *Proceedings of the 64th International Astronautical Congress (IAC)*, 2013.

[32] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal War in Memory," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 48–62.

[33] S. Crane, C. Liebchen, A. Homescu, L. Davi, P. Larsen, A.-R. Sadeghi, S. Brunthaler, and M. Franz, "Readactor: Practical code randomization resilient to memory disclosure," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 763–780.

[34] K. Pietrzak, "A leakage-resilient mode of operation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009, pp. 462–482.

[35] J. Finke, R. Thummala, R. Elbasheer, P. Hansen, W. Henry, D. Mamula, A. Noor, T. York, K. Zheng, and G. Falco, "Satellite cybersecurity testbed to improve commercial space security," 10 2023.

[36] M. Bozdal, M. Samie, and I. Jennions, "A survey on can bus protocol: Attacks, challenges, and potential solutions," in *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE, 2018, pp. 201–205.

[37] O. Avatefipour and H. Malik, "State-of-the-art survey on in-vehicle network communication (CAN-Bus) security and vulnerabilities," *arXiv preprint arXiv:1802.01725*, 2018.

[38] L. Dariz, M. Selvatici, M. Ruggeri, G. Costantino, and F. Martinelli, "Trade-off analysis of safety and security in can bus communication," in *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. IEEE, 2017, pp. 226–231.

[39] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical dos attacks on embedded networks in commercial vehicles," in *Information Systems Security: 12th International Conference, ICISS 2016, Jaipur, India, December 16-20, 2016, Proceedings 12*. Springer, 2016, pp. 23–42.

[40] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–40, 2009.

[41] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 143–158.

[42] D. Sabogal and A. D. George, "Towards resilient spaceflight systems with virtualization," in *2018 IEEE Aerospace Conference*, 2018, pp. 1–8.

[43] D. Koisser, B. Ferdinand, P. Jauernig, E. Stapf, M. Wallum, D. Fischer, and A.-R. Sadeghi, "Hardware-based isolation for advanced safety and security in spacecraft," in *Proceedings of the International Conference on Space Operations*, 2023.

[44] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 479–498.

[45] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 1–14.

[46] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," in *Proceedings of the 52nd annual design automation conference*, 2015, pp. 1–6.

[47] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE, 2004, pp. 272–282.

[48] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. Van Doorn, and P. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *Proceedings of the twentieth ACM symposium on Operating systems principles*, 2005, pp. 1–16.

[49] Y. Li, J. M. McCune, and A. Perrig, "Viper: Verifying the integrity of peripherals' firmware," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 3–16.

[50] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: secure and minimal architecture for (establishing dynamic) root of trust." in *Ndss*, vol. 12, 2012, pp. 1–15.

[51] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A Verified Hardware/Software Co-Design for Remote Attestation," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1429–1446.

[52] I. De Oliveira Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik, "On the toctou problem in remote attestation," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2921–2936.

[53] F. Arkannezhad, J. Feng, and N. Sehatbakhsh, "IDA: Hybrid Attestation with Support for Interrupts and TOCTOU," in *NDSS*, 2024.

[54] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 743–754.

[55] Z. Sun, B. Feng, L. Lu, and S. Jha, "OAT: Attesting operation integrity of embedded devices," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1433–1449.

[56] N. Yadav and V. Ganapathy, "Whole-program control-flow path attestation," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2680–2694.

[57] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[58] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A.-R. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 384–391.

[59] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "Litehax: lightweight hardware-assisted attestation of program execution," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[60] Y. Zhou, X. Wang, Y. Chen, and Z. Wang, "Armlock: Hardware-based fault isolation for arm," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 558–569.

[61] Y. Chen, S. Reymondjohnson, Z. Sun, and L. Lu, "Shreds: Fine-grained execution units with private memory," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 56–71.

[62] A. Vahldiek-Oberwagner, E. Elnikety, N. O. Duarte, M. Sammler, P. Druschel, and D. Garg, "ERIM: Secure, Efficient In-process Isolation with Protection Keys (MPK)," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1221–1238.

[63] A. Voulimeneas, J. Vinck, R. Mechelinck, and S. Volckaert, "You shall not (by)pass! practical, secure, and fast PKU-based sandboxing," in *European Conference on Computer Systems (EuroSys)*, 2022.

[64] D. Schrammel, S. Weiser, S. Steinegger, M. Schwarzl, M. Schwarz, S. Mangard, and D. Gruss, "Donky: Domain Keys–Efficient {In-Process} Isolation for RISC-V and x86," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1677–1694.

[65] K. Dinh Duy, K. Cho, T. Noh, and H. Lee, "Capacity: Cryptographically-enforced in-process capabilities for modern arm architectures," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 874–888.

[66] M. Hedayati, S. Gravani, E. Johnson, J. Criswell, M. L. Scott, K. Shen, and M. Marty, "Hodor: Intra-Process isolation for High-Throughput data plane libraries," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 489–504. [Online]. Available: http://www.usenix.org/conference/atc19/presentation/hedayati-hodor

[67] Q. Wang and S. Sawhney, "Vecure: A practical security framework to protect the can bus of vehicles," in *2014 International Conference on the Internet of Things (IOT)*. IEEE, 2014, pp. 13–18.

[68] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, "Intrusion detection system for automotive controller area network (can) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, pp. 1–17, 2019.

[69] S. Abbott-McCune and L. A. Shay, "Intrusion prevention system of automotive network can bus," in *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2016, pp. 1–8.

[70] D. Yu, R.-H. Hsu, J. Lee, and S. Lee, "EC-SVC: Secure can bus in-vehicle communications with fine-grained access control based on edge computing," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1388–1403, 2022.

[71] Ú. Erlingsson, "Low-level software security: Attacks and defenses," in *International School on Foundations of Security Analysis and Design*. Springer, 2006, pp. 92–134.

[72] J. Ligatti, M. Abadi, M. Bidiu, and U. Erlingsson, "Control flow integrity," in *Proceedings of the 12th ACM Conference on Computer and communications security*, 2005.

[73] M. Zhang and R. Sekar, "Control flow and code integrity for cots binaries: An effective defense against real-world rop attacks," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 91–100.

[74] M. Payer, A. Barresi, and T. R. Gross, "Fine-grained control-flow integrity through binary hardening," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings 12*. Springer, 2015, pp. 144–164.

[75] P. Muntean, M. Neumayer, Z. Lin, G. Tan, J. Grossklags, and C. Eckert, "Analyzing control flow integrity with llvm-cfi," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 584–597.

[76] B. Niu and G. Tan, "Modular control-flow integrity," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 577–587.

[77] ——, "Per-input control-flow integrity," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 914–926.

[78] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, "Softbound: Highly compatible and complete spatial safety for c," in *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2009.

[79] L. Davi, P. Koeberl, and A.-R. Sadeghi, "Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[80] H. Liljestrand, T. Nyman, K. Wang, C. C. Perez, J.-E. Ekberg, and N. Asokan, "PAC it up: Towards pointer integrity using ARM pointer authentication," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 177–194.

[81] N. Zeldovich, H. Kannan, M. Dalton, and C. Kozyrakis, "Hardware Enforcement of Application Security Policies Using Tagged Memory," in *OSDI*, vol. 8, 2008, pp. 225–240.

[82] X. Zhou, J. Li, W. Zhang, Y. Zhou, W. Shen, and K. Ren, "Opec: operation-based security isolation for bare-metal embedded systems," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 317–333.

[83] S. Park, H. Kim, S.-Y. Kang, C. H. Koo, and H. Joe, "Lua-based virtual machine platform for spacecraft on-board control software," in *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*, 2015, pp. 44–51.

[84] H. Joe, H. Jeong, Y. Yoon, H. Kim, S. Han, and H.-W. Jin, "Full virtualizing micro hypervisor for spacecraft flight computer," in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, 2012.

[85] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[86] V. Kuznetzov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-pointer integrity," in *The Continuing Arms Race: Code-Reuse Attacks and Defenses*, 2018, pp. 81–116.

[87] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *Proceedings of the fourteenth ACM symposium on Operating systems principles*, 1993, pp. 203–216.

[88] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 90–102.