Space Odyssey: An Experimental Software Security Analysis of Satellites

Johannes Willbold*, Moritz Schloegel*‡, Manuel Vögele*, Maximilian Gerhardt*,
Thorsten Holz‡, Ali Abbasi‡

*Ruhr University Bochum, firstname.lastname@rub.de
‡CISPA Helmholtz Center for Information Security, lastname@cispa.de

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10351029

***Abstract*—Satellites are an essential aspect of our modern society and have contributed significantly to the way we live today, most notable through modern telecommunications, global positioning, and Earth observation. In recent years, and especially in the wake of the *New Space Era*, the number of satellite deployments has seen explosive growth. Despite its critical importance, little academic research has been conducted on satellite security and, in particular, on the security of onboard firmware. This lack likely stems from by now outdated assumptions on achieving security by obscurity, effectively preventing meaningful research on satellite firmware.**

In this paper, we first provide a taxonomy of threats against satellite firmware. We then conduct an experimental security analysis of three real-world satellite firmware images. We base our analysis on a set of real-world attacker models and find several security-critical vulnerabilities in all analyzed firmware images. The results of our experimental security assessment show that modern in-orbit satellites suffer from different software security vulnerabilities and often a lack of proper access protection mechanisms. They also underline the need to overcome prevailing but obsolete assumptions. To substantiate our observations, we also performed a survey of 19 professional satellite developers to obtain a comprehensive picture of the satellite security landscape.

***Index Terms*—satellites, satellite security, space segment, satellite firmware, threat taxonomy, software security**

1. Introduction

Satellites are sophisticated technical devices that are placed in outer space for research purposes or to provide terrestrial applications with services that leverage the coverage of the Earth's surface. While the first satellite, *Sputnik*, dates back to 1957, we are in the midst of a renaissance of spaceflight referred to as *New Space Era* [1]. Especially in recent years, we have observed an enormous growth in the number of earth-orbiting satellites. According to the United Nations Office for Outer Space Affairs (UNOOSA), their number has almost doubled from $4,867$ in 2019 to $9,350$ in 2022 [2]. The vast majority of these satellites form mega-constellations like *Starlink*, which plans to launch more than $40,000$ satellites in the coming years [3].

Small satellites [4] are at the heart of this *New Space Era* as their size and the widespread use of Commercial off-the-shelf (COTS) components makes them affordable even for small institutions. Furthermore, they cover a broad spectrum of use cases ranging from commercial applications (like Earth observation, machine-to-machine communication, and Internet services) to research applications, such as technology testing, weather and earthquake forecasting, and even interplanetary missions [5]–[8].

Although their applications vary widely, small satellites commonly consist of radio equipment and microcontroller boards. Hence—in the broadest sense—they are computer systems connected to a ground station on Earth and, sometimes, even to other satellites. Because they rely on wireless connections for command and control and use microcontrollers, they are potentially as vulnerable to attacks as any other connected IT platform on Earth.

This issue has not been very relevant in the past, since access to ground stations was expensive and limited to large satellite operators. However, the situation changed fundamentally in recent years. Nowadays, ground stations are even affordable for private individuals and with the emergence of Ground Station as a Service (GSaaS) models, such as those offered by *Amazon Web Services* [9] and *Microsoft Azure* [10], the entry barrier becomes even lower. We have seen in the mobile network security domain how the providers' assumption that the radio equipment required for attacks would be too costly and out of reach for attackers was ultimately disproved by technological advances [11]. Similarly, affordable ground stations create a novel attack surface, where adversaries can communicate with satellites and take advantage of software vulnerabilities. If they successfully compromise the satellite's firmware, they can access the satellite and potentially take complete control of the system.

Despite warnings being made early [12], little has been done to address this problem for several reasons, as Falco [13] points out. While the lack of security standards

DOI 10.1109/SP46215.2023.00131

for satellites and the complex supply chain complicate the situation, the main reason is the inaccessibility of satellite firmware. Historically, satellite developers have relied on *security by obscurity*. The developers of the *Iridium* network even mentioned that their system would be too complex for attackers [13]. Still, Driessen et al. have shown that attackers can successfully decrypt the communication of the network [14]. Especially the inaccessibility of satellites in orbit makes dumping of the firmware by researchers very challenging (if not impossible), impeding progress in this area. Hence, the developers of satellite firmware act as *gatekeepers* and do not provide researchers with research subjects. Notably, Pavur and Martinovic [15], as well as Falco [13], acknowledge that the topic is still understudied and conclude that collaboration between satellite development and the security field is required. Additionally, well-known topics like the security of satellite communication, the security of satellite-based Internet services, and threat scenarios for satellites have recently gained increasing attention [16], [17]. However, discussions around individual satellites typically lack technical details of satellite and real-world foundations due to the inaccessibility of satellite software.

In this paper, we make three contributions to systematically improve satellite security. First, we present a taxonomy of threats against onboard satellite firmware. Such a systematic review of the attack surfaces allows us to better represent the complex nature of satellites and categorize security-relevant findings throughout the paper.

Second, we conduct an experimental and comprehensive security analysis of *three* real-world, in-orbit satellites to better understand the attack surface and the current state of software security in this particular domain. We focus on Low Earth Orbit (LEO) satellites, as this orbit is the main focus of the *New Space Era*. The most prevalent satellite class is the *nanosatellite*, more specifically, the *CubeSat* which is a standard form factor of $10\,\mathrm{cm}$ cubes (called *Units* or *U*). These satellites typically weigh less than $1.33\,\mathrm{kg}$ per U and are used in many different projects. After a long period of persuasion, trust building, discussions, and contracts, we obtained access to several satellite firmware images that we were able to analyze. As a result of our security assessment, we found six different kinds of security vulnerabilities in recently launched modern spacecraft, including unprotected telecommand interfaces. All vulnerabilities have been responsibly disclosed to the vendors. Note that the entry barrier to identify these vulnerabilities was complex, given the sensitive nature of these systems. To the best of our knowledge, our work is the first to demonstrate exploitation of satellite firmware vulnerabilities allowing attackers to gain persistent control over the satellite.

Third, we conducted a survey of 19 professional satellite engineers and developers to broaden the scope of our research. In total, the responses cover technical information on 17 satellites, and the participants worked on a total of 132 satellites. Our survey reveals that protocol obscurity is as prevalent as encryption for access protection and that small development teams are rather inclined to develop full custom protocols instead of using existing ones. As one of our survey participants mentioned: "*We focused on providing a functioning system instead of a secure one*".

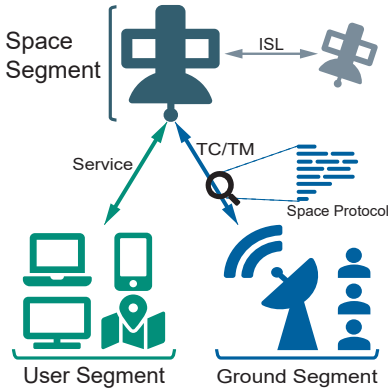


Figure 1: General overview of satellite operations, which evolve around the *space segment* and the *ground segment* to provide service to the *user segment*.

In summary, we make the following main contributions:

- A *taxonomy of threats* and accompanying *attacker models* against onboard satellite firmware that provides a systematic overview and enables us to derive satellite-specific threat models.
- A *systematic security analysis* of three real-world satellite firmware images that uncovered 13 vulnerabilities and is based on an attacker model accounting for recent technical developments (e.g., GSaaS).
- A *satellite community survey* to challenge our technical results and shed light on the views that professionals from the space community have on security.

2. Satellite Context

An *artificial satellite* (typically abbreviated as *satellite*) is an object intentionally placed in outer space that orbits another body, such as the Earth. Satellites are designed to operate in the harsh conditions of space, which include extreme temperature variations of about $200\,^{\circ}\mathrm{C}$ occurring more than ten times a day, a near vacuum, and cosmic radiation. However, deployment in space is often necessary to provide a space-derived service to Earth, with common satellite applications including communications, Earth observation, and research. While most satellites are deployed in LEO (250–$2000\,\mathrm{km}$), other orbits such as the Geostationary Orbit (GEO) ($35{,}786\,\mathrm{km}$) may be necessary depending on the purpose of the satellite.

2.1. Satellite Operations

Satellite operations, as shown in Figure 1, evolve around three main components: the *group segment*, which operates the satellite-based service, *space segment* consisting of all space assets, and the *user segment*, which receives a satellite

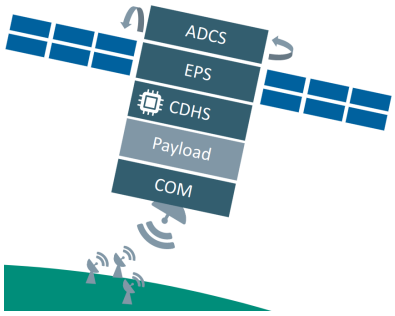


Figure 2: Overview of the different components typically found in a common satellite that features default components for a satellite payload (light grey) and bus (dark grey).

service such as Global Positioning System (GPS) or communication.

2.1.1. Ground Segment. The *ground segment* is the center of all satellite operations throughout the entire lifetime of a satellite. A team of *operators* communicates with the satellite using a Ground Station (GS) to provide new instructions to the satellite, referred to as Telecommand (TC). In turn, the satellite sends Telemetry (TM) back to the GS, providing information about the satellite's status, errors, and other metrics. The TC uses a *space protocol*, which we describe in Section 2.3. In the following, we refer to the combination of TC and TM data as *TC/TM traffic*.

2.1.2. Space Segment. The *space segment* includes all spacecraft involved in the satellite operations, which may be just a single satellite or an entire constellation. These satellites are initially launched into orbit using a *launch vehicle*, i.e., a rocket, and then undergo an *orbital deployment phase* to initiate communications with the *ground segment*. During the *nominal operations phase*, which describes regular service operations, the satellites may communicate with each other via an Inter-Satellite Link (ISL).

2.1.3. User Segment. A terminal, e.g., a Very Small Aperture Terminal (VSAT) or GPS receiver in the *user segment*, receives the service provided by the *space segment*. Note that some satellites, like Earth observation satellites, communicate exclusively with their *ground segment* and do not include a *user segment*.

2.2. Satellite Architecture

Figure 2 introduces the components commonly found in a modern satellite, which are split into a *satellite payload* and a *satellite bus*. The *satellite payload* consists of mission-specific equipment, such as a high-resolution camera for Earth observation or powerful radio hardware for telecommunications. The *satellite bus* encompasses all components required to operate and maintain the satellite. It is designed to operate independently of the payload, but conversely, the payload relies on the bus. We refer to this separation as *bus-payload separation*. In our security analysis, we therefore focus on the satellite bus because, unlike many payloads, it grants attackers full control over a satellite.

2.2.1. Command and Data Handling System (CDHS). The bus is centered around the CDHS, which manages the satellite and controls all functions of the spacecraft. The CDHS uses an On-Board Controller (OBC) that employs a computing platform, i.e., a microcontroller and memory, similar to terrestrial embedded devices. The software executed on this system is called the On-Board Software (OBSW), which is the main focus of our work. It implements a remote-control server, usually based on a Real-Time Operating System (RTOS), akin to terrestrial real-time applications. The main task of the OBSW is handling TC/TM traffic, providing data storage, scheduling commands, performing autonomous actions, and updating the program code. Crucially, the OBSW, as any software, is vulnerable to common software faults that attackers may exploit to gain unauthorized control over the CDHS.

2.2.2. Communications Module (COM). Communication with the GS is handled by the Communication Module (COM), which consists of an antenna, a radio, and sometimes a computing setup to handle decoding, protocol implementations, and access projection. Further, the COM is usually only dedicated to TC/TM traffic, whereas high bandwidth traffic, such as the payload data downlink, is often handled by a more powerful radio setup. Since the COM is directly coupled with the CDHS to handle TC/TM traffic, it is also the main entry point for adversaries, creating a significant attack surface. Additionally, protocol implementations in the COM may be security relevant, as it may be the first line of defense.

2.2.3. Attitude Determination and Control System (ADCS). Satellites employ an ADCS to determine and adjust their attitude so that they can point antennas towards the Earth and solar panels at the Sun. In addition, the ADCS performs autonomous *detumbling* to stop the satellite's angular spinning after it is released from the launch vehicle, which is necessary to establish the initial link. Additionally, the satellite might also use a *thruster* to create an Attitude and Orbit Control System (AOCS) for minor orbit changes, which is particularly important for security reasons. An AOCS makes satellites *cyber-physical systems* as they can affect their physical environment by crashing into another object, which can lead to a devastating orbital chain reaction (cf. Section 3.1.1).

2.2.4. Power Supply (EPS). The Electrical Power System (EPS) is the satellite's power supply, usually generated by solar panels and stored in batteries to provide power in the absence of light, such as when circling around the Earth. Additionally, it is critical that the battery never fully depletes, as the satellite cannot restart in that case. Hence, power management is crucial to survivability, as battery deep-draining is of interest to attackers to permanently disable a satellite.

2.2.5. Payload. While the *payload* mostly deploys mission-specific equipment, it often also deploys a Payload Data Handling System (PDHS) that acts akin to a CDHS. A PDHS can either receive control traffic from a Payload Communication Module (PLCOM), which can be any receiver on the *payload*, or it can handle general data processing tasks from the payload equipment. Due to the high degree of customization on the payload, the exact terms for the PDHS and PLCOM may vary in other satellite descriptions.

2.3. Satellite Communication Protocols

A satellite's TC/TM traffic is communicated via a satellite communications protocol, which we referred to as *space protocol* in Figure 1. The main organization for publishing such space protocols is the Consultative Committee for Space Data Systems (CCSDS), a consortium of numerous space agencies that agree on standards. Ultimately, the CCSDS provides a wealth of protocol standards for communicating with all components and parties involved in spacecraft operations [18]. These standards cover all layers of the *OSI model*, and there are usually several options per layer [19]. The protocols mentioned in this work are the Space Data Link Security (SDLS) for the *data link layer*, which also implements a security extension [20], and the Space Packet Protocol (SPP) for the *network layer*. Note that the CCSDS collection is more comparable to the collection of all network protocols commonly used on the Internet rather than a specific protocol. In the following, we refer to the collection of CCSDS protocols as *CCSDS protocol*.

3. Satellite Firmware Threats

We now propose a taxonomy of firmware threats against satellites as a three-layer representation shown in Figure 3. The figure summarizes our satellite firmware threat insights to provide a compact and functional overview. Previous work by Falco and Boschetti [21] provides a broad taxonomy of general threats against satellites, including environmental, physical, and *digital-cyber-technical* risks. The latter serves as our high-level *attacker goals* such that our works integrate well with their taxonomy. Note that their work provides a more abstract, high-level taxonomy, while ours focuses on detailed technical threats to satellite firmware.

3.1. Threat Taxonomy

Our taxonomy, visualized in Figure 3, includes three layers, which we describe using a *top-down* approach. On the highest layer, the *control layer*, we find the ultimate attacker goals. To achieve them, the attacker must target some component that represents a functional satellite component (cf. Section 2.2) on the *components layer*. To communicate with a component, an attacker must first access one of the interfaces that reside on the *interface layer* and manage interactions between components and external actors, such as the GS.

We use solid lines to describe the *hierarchy* of elements (i.e., the *bus* is part of the satellite, and the CDHS is part of the *bus*, cf. Figure 3) and a set of dot-lined arrows to describe *attacks paths* (i.e., an attacker has to access the COM before issuing a telecommand in the CDHS). We use colors corresponding to the different layers of Figure 3.

3.1.1. Control Layer. We model the high-level attacker goals against the Satellite based on the final digital-cyber-technical risks Falco and Boschetti identified [21]. Then, we identify intermediate goals an attacker must achieve towards their final goal and differentiate between two target components, the Bus and the Payload. Recall that the bus controls the payload, thus allowing to achieve goals on the payload from the bus, but not vice-versa. This separation of bus and payload is commonly found in satellites, originating from safety concerns to prevent payload equipment faults from propagating. In the following, we elaborate the *attacker goals* and how they concern the *bus-payload separation*.

Denial of Service/Control. Denial of Service (DoS) is the most common attack vector on satellites today [22] and threatens a satellite's *availability*. It can be achieved both on the bus and payload, which deploys the equipment for the satellite's service. Contrary, a denial of *control* can only be achieved through the bus.

Malicious Data Interaction. Attackers may want to extract or manipulate satellite data targeting the bus or payload. *Control data interaction* concerns flight-critical data and access protection secrets on the *bus*, requiring a compromised *bus*. *Payload data*, on the other hand, can be accessed from the *payload*, e.g., data from cameras or other experiments in the payload.

Seizure of Control. Last, attackers may want to seize full control over the satellite. Usually, the TCs offered by the bus are not sufficient to execute arbitrary actions but rather provide a set of pre-defined interactions, prompting the need to find a vulnerability granting *arbitrary code execution* on the bus.

A seizure of control is not only problematic for the satellite's owners, but has potentially devastating consequences on the entire space ecosystem. If the satellite deploys thrusters, attackers could attempt to invoke the *Kessler Syndrome*, an effect in which the debris from one satellite crash collide with other satellites, destroying them and emitting new debris, resulting in a chain reaction. These debris could potentially make space inaccessible for decades, as shown in simulations [23], [24]. These potential consequences of a single successful satellite hack are largely ignored by the security community, even though they could heavily affect spaceflight as we know it.

3.1.2. Components Layer. The *components layer* consists of the relevant common satellite components (cf. Section 2.2), the *Bus-Payload Link*, and the Untrusted Data Handling System (UDHS). In practice, these components can be separate hardware components or be combined in a single firmware image. Here, we divide them by function.

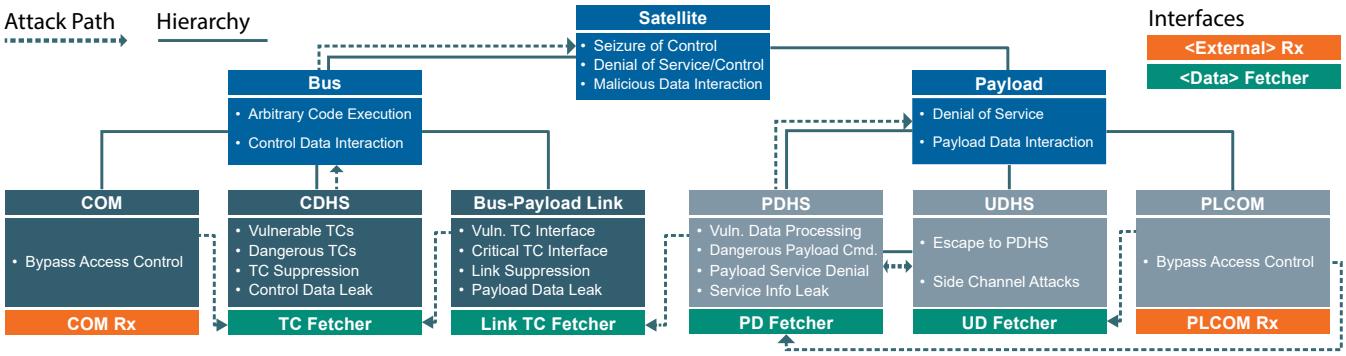


Figure 3: A taxonomy of threats against satellite firmware

In the following, we discuss each component, its task, and the threats against it. We categorize threats by the three well-known pillars of security – Integrity (1), Availability (3), and Confidentiality (4). Additionally, we consider *Stability (2)* as a sub-category of *Integrity* to describe threats that exist *by design* in a satellite but risk the satellite's operational integrity if accessible to attackers.

COM / PLCOM. The COM receives incoming TCs over a remote channel (i.e., a radio), while the PLCOM either receives payload data or TCs intended for the payload. As we consider only attacks against the satellite firmware, we summarize threats against the (PL)COM as *bypassing access control*. This includes general radio frequency threats such as *Man-in-the-Middle* attacks and cryptographic threats like *timing side-channels*, which previous work explored in detail [25], [26]. Additionally, this includes threats targeting the microcode demodulating or decoding TCs.

CDHS. The CDHS handles incoming TCs by executing the associated function in the satellite's firmware that performs a direct or scheduled action on the satellite (cf. Section 2.2). It must fulfill the following requirements:

(1) Integrity: *Vulnerable TCs* can allow an attacker to access private data, hijack the control flow, or leak information. This also includes all memory corruption vulnerabilities in handling TCs, e.g., buffer overflows, format string vulnerabilities, or use-after-free vulnerabilities [27].

(2) Stability: Some overly permissive *dangerous TCs* can enable an attacker to take over the satellite's access control, firmware update mechanism, control flow, or critical data simply by issuing the respective TC. Such TCs often exist for debugging purposes. TCs that provide arbitrary memory write gadgets (e.g., for debugging) or arbitrary changes to access control secrets should not be implemented, while access to the firmware image should require an additional layer of verification (e.g., signed images), which is generally advisable and called *defense in depth* [28].

(3) Availability: The handling of TCs must be available at all times for time-critical actions such as course corrections. We summarize all threats against the CDHS availability as *TC suppression* threats as they suppress the satellite's ability to handle TCs.

(4) Confidentiality: TCs may allow an attacker to leak secrets, e.g., concerning access control, and thereby compromise the entire satellite if this leak allows elevated access. We summarize *control data leaks* as threats against the CDHS' confidentiality.

Bus-Payload Link. The *Bus-Payload Link* provides a bridge for the payload to interact with the bus. This is necessary as the payload may need access to monitoring and control capabilities of the bus to control the payload. These functions vary depending on the mission and payload, but may include options to toggle power to payload components. Hence, the link provides an API-like surface between different layers of trust.

(1) Integrity: The link does not deploy own TCs but uses the ones from the CDHS through *TC interfaces*. These API-like interfaces can be vulnerable akin to *vulnerable TCs*, which we classify as *vulnerable TC interfaces*.

(2) Stability: *Critical TC Interfaces* can compromise the *bus* by deliberately offering overly permissive command functionality to the (potentially compromised) payload. Critical TCs include unlimited power management or adjusting the satellite's attitude to make radio communication impossible. Commands issued by the payload should only impact payload equipment and must be reversible at any point by the bus. The difference between *critical* and *dangerous* TCs is that *critical* TCs offer functionalities that should be exclusive to the bus while *dangerous* TCs should either not exist at all (even on the bus) or must require additional authentication/verification.

(3) Availability: If more than one PDHS or UDHS exists, none of them should be able to deny the link's availability to the others. As such, we identify *link suppression* as threat to the link between connected payload components.

(4) Confidentiality: The link is compromised if payload components can extract information intended exclusively for other payload components. We summarize these as *payload data leaks*.

PDHS. The PDHS is the payload's data processing system and, depending on the missions, acts as a payload command system with packets from the PLCOM. As such, the system exercises immediate control over the payload functionalities or is involved in processing untrusted data from the *user segment* (cf. Section 2.1). In general, as the PDHS can deploy any computing task, especially when processing untrusted payload user traffic, the threat scenario

shifts towards common *system security threats*, leaving the following categories intentionally vague.

(1) Integrity: We summarize all classical *software vulnerability threats* akin to *vulnerable TCs* as *vulnerable data processing*.

(2) Stability: This is only a concern if the PDHS handles command traffic similar to the CDHS for the payload. In this case, we identify the *dangerous payload command* threat category, similar to dangerous TCs.

(3) Availability: The PDHS availability is threatened if the ability to process incoming packets is inhibited, leading to a denial of the payload service. Hence, we call this a *payload service denial*.

(4) Confidentiality: We identify general *service information leaks* as threats against the PDHS' confidentiality.

UDHS. The UDHS runs untrusted code of payload users directly on the satellite. Since the code is untrusted, it must be isolated from regular payload operations on the PDHS. This component is not part of the common satellite architecture (cf. Section 2.2), but with the trend towards renting satellite capabilities and considerations to build orbital cloud computing services [29], [30], it is time to consider this component. In practice, we found this component deployed in our case studies in Section 5.2.

The PDHS and UDHS have a special relationship in our taxonomy. First, an UDHS can be part of a PDHS (i.e., the PDHS runs an operating system, where one isolated process runs untrusted code), where the same environment also runs the payload data processing. Second, a PDHS can be part of a UDHS, i.e., the UDHS deploys an (untrusted) application that processes data from receiving components. From an attacker's perspective, the parsing application deployed by the UDHS acts as PDHS. We call this UDHS-wrapped PDHS the UDHS-PDHS. From an attack path perspective, the UDHS-PDHS is still isolated, making it different from the main PDHS while facing the same threats as the PDHS. We highlight an example of this mechanism in Section 5.2.

(1) Integrity + (2) Stability: The UDHS' integrity is threatened if the environment isolation is attacked, which we call *Escape to PDHS* threats.

(3) Availability: We do not consider UDHS availability threats, as only a UDHS-PDHS has availability obligations.

(4) Confidentiality: Extracting information from the isolated host environment threatens the UDHS' confidentiality, which we summarize as *side channel attacks*.

3.1.3. Interface Layer. *Interfaces* form the third and lowest layer of our taxonomy. Whenever a component interacts which another component or an external source, an interface is used in between. We distinguish between two types of interfaces, *external interfaces* that interface a component with an external element (i.e., the GS) and *internal interfaces* that act as data interfaces and, hence, are called *Data Fetchers*, where we replace the word *Data* with a more precise description like TC if applicable. Every *interface* has exactly one parent, e.g., the parent of *COM Rx* is *COM*. However, a component may have multiple interfaces, e.g., the CDHS may have two different *TC fetchers* for the COM and *Bus-Payload Link*. In Figure 3, we omit the arrows between *interfaces* and *components* for simplicity, but every *interface* has a hierarchy line to the parent and a dot-lined attack path arrow from the interface to the *component*.

External Rx. External interfaces receive data from outside the satellite (i.e., radios or optical receivers). We omit threat considerations since, in our model, these interfaces only implement purely hardware-based operations without software and would only be subject to electromagnetic and radio frequency threats. If the satellite uses firmware, i.e., in the form of microcodes to perform signal demodulation, we consider this to be part of the COM as exploitation might potentially yield a bypass to access control. Hence, we only consider this interface to model our attackers, as we detail in Section 4. In addition, we solely consider receivers of structured data that some component parses. For example, scientific equipment to measure radiation or a thermometer cannot receive bit-exact structured data, excluding them from our considerations.

Data Fetcher. Internal interfaces manage the interactions between two *components*. We call these interfaces *data fetchers* as they internally fetch data from one component and provide it to their parent. Since they only *receive* traffic from components and forward it to their parent, they only face *(1) Integrity* and *(3) Availability* threats. No distinction between stability and integrity can be made, and compromising *confidentiality* requires a return channel.

(1) Integrity + (2) Stability: An attacker may desire to manipulate existing data by injecting new or altering existing data. As such, *data injection* and *data alteration* are threats to the integrity of data fetchers. For example, the *TC Fetcher* interface is compromised if an attacker manages to inject an additional TC, although only one was sent.

(3) Availability: An attacker may compromise the interface's ability to pass all incoming data packets to its parent. Since memory is often limited, these interfaces usually use a *ring-buffer* that swaps out older packets for newer packets. By flooding the buffer and continuously rotating not-yet-processed packets out, the interface is compromised (*data flooding* threat). In addition, the data fetcher's forwarding ability can be inhibited in some way (other than overwhelming), which we call *forward suppression* threat.

3.2. Deriving Satellite-specific Insights

Our taxonomy in Figure 3 highlights not only threats but also attack paths and all computing components found on a satellite. Hence, our taxonomy is functional and allows us to derive satellite-specific models that enumerate all possible attack trees and the full attack surface. In the following, we first describe how to derive a satellite-specific model, which we also use for all three case studies (cf. Section 5). Then, we explain how all attack trees and attacker surfaces can be extracted from this model.

3.2.1. Deriving a Satellite-specific Model. Before a satellite-specific model (subsequently abbreviated as *model*) can be derived from our taxonomy for a specific satellite, we

stress that a crucial prerequisite is a sufficient understanding of the satellite's internals (either through documentation or reverse engineering). Then, the model can be derived in two steps: First, we match the components to the actual satellite components. Second, we match the interfaces between these components to model the real-world interactions of the components in the satellite.

In detail, when matching components, each component in the *component layer* can be duplicated or removed to match the concrete satellite requirements. For example, if a satellite does not have a PDHS or UDHS, both (as well as the *Bus-Payload Link*) can be removed, leaving only the COM and CDHS. If the satellite has multiple PLCOM components, we duplicate the PLCOM component while keeping the *hierarchy line* and *attack path arrow* for each individual COM. Notably, no new components can be created, only existing taxonomy components can be duplicated.

Next, the *interfaces* are matched to the components. Each component initially comes with *its own* interface (cf. Figure 3). However, as each interface can only have a single parent but multiple children, multiple interfaces can be shared as receiving interfaces for a single component. For example, the CDHS can use the same *TC Fetcher* for multiple COMs, i.e., if there are active redundant COMs.

3.2.2. Extracting the Attack Tree and Surface. After deriving this model, the dotted *attack path* arrows form a directed graph. In Section 4, we discuss our attackers in accordance with our interfaces, where each attacker is *connected* to an interface, making these *attacker-connected interfaces*. The combination of all attacker-connected interfaces forms the *attack surface*. By following all possible paths from an attacker-connected interface to an element of the control layer, we can extract all possible attack paths, where each path is a subgraph of our *attack tree*.

In our case studies, we use this attack tree representation as it provides a more intuitive representation of the security aspects for a specific satellite.

4. Attacker Model

Using our taxonomy of threats against satellite firmware, we formulate four attacker models, accounting both for the attacker's *knowledge* and *level of access*.

4.1. Attacker Knowledge

In a first step, we review a prevailing but outdated assumption that needs to be revised.

Security by Obscurity. For decades, the satellite community and developers have acted as *gatekeepers* for the topic of satellite security [15]. By keeping the software and components of satellites under lock, they created a "barrier of obscurity" that prevented any meaningful research on this subject. Hence, external communities had no way to study satellite internals and potential security issues.

In recent years, this changed as the developments in the space domain have moved towards COTS components [15], [31], open satellite designs [7], [32], and open-source libraries [33]. These factors have been multiplied by the explosive growth in the number of satellites [4] and the inherent increase in the size of the community. Hence, the number of people holding knowledge about satellites steadily increases. Overall, we argue that a transformation is slowly happening concerning the effectiveness of security by obscurity in space-borne assets.

Revised Assumption. As a result, we must assume that attackers have detailed knowledge of the *target satellite*, including detailed documentation and access to firmware images. Further, several open-source satellites already enable attackers to study satellites [34]–[36]. We therefore assume attackers have detailed knowledge of satellites, including their firmware, except for cryptographic secrets.

4.2. Attacker Access Level

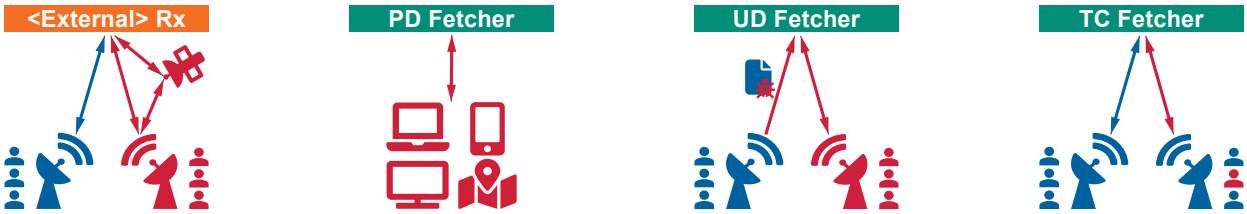
We also adjust a common misconception regarding the attacker's access level to the space segment.

Myth of Inaccessibility. Until recently, it was generally assumed that satellites always communicate with prohibitively expensive GSs. As a result, only few actors could attack a satellite (similar to the assumption for mobile cell phone networks many years ago). This assumption had a major impact on the adaptation of security features in satellites [15]. However, GS prices have dropped significantly in the past few years. Today, it is possible to create a fully functional GS for less than $10k [37], and there are open-source communities around developing GSs [38]. In addition, GSaaS providers such as *Amazon Web Services* or *Microsoft Azure* rent a GS to the user [9], [10] or allow GS owners to monetize unused GS capacity by temporarily renting it to end users. As a result, one does not even need to own GS equipment to interact with satellites. Additionally, transceivers for specific satellite services have become so compact and cheap that they can be found in consumer electronics, such as the *iPhone 14* [39]. Furthermore, there are now many LEO satellite constellations in space with satellite-to-satellite communication capability. At the same time, there is an increasing number of smaller research LEO satellites. There are already a number of satellites with significant communication capabilities in space that are even intended to be used by third parties [32].

Revised Assumption. Therefore, we believe that there is a paradigm shift in the assumption that satellites are inaccessible, which is particularly pronounced for LEO satellites. Consequently, we divide attackers into *external attackers*, *payload users*, *payload service hosters*, and *operators* based on their association with the satellite.

4.3. Attacker Models

In the following, we describe the attacker models while considering our revised assumptions. In addition, we connect each attacker to an *interface* from our taxonomy presented in Section 3.1.3.



(a) *External Attackers* communicate with the satellite using a custom GS and ISL

(b) *Payload Service Attackers* interact with the satellite's payload data handling

(c) *Malicious Service Hosters* execute untrusted code on the isolated UDHS

(d) *Semi-Privileged Insiders* use elevated privileges to issue non-critical TCs

Figure 4: Our attacker models vary in their ability to interact with the target satellite to use all potential satellite interactions.

4.3.1. External Attackers. An external attacker, as shown in Figure 4a, can use a custom GS or a custom satellite to interact with the target satellite. As such, external attackers can send arbitrary traffic to any interface receiving external traffic while also receiving any response. Hence, in our taxonomy, external attackers can interact with every External Rx interface. We can distinguish between two types of external attackers.

Custom Ground Station Attacker. Communication with the GS is a satellite's primary *command-and-control channel*. Even if this channel is protected via access control mechanisms, an attacker may be able to *bypass access control*, which we identified as the primary threat to *COM* components (cf. Section 3.1.2). To this end, we propose a *custom GS attacker* that can communicate with the target satellite through any GS except the one used by the satellite operators, as it may contain access control secrets. We further assume that attackers have the required knowledge to establish a radio connection, i.e., frequencies, modulations, and orbital position, except for the aforementioned secrets.

ISL Attacker. In addition to custom GS access, we assume that an external attacker has access to custom satellites to communicate with *external receiving interfaces* over ISL connections (cf. Section 2.1).

4.3.2. Malicious Payload Users. Payload users are actors of the *user segment* (cf. Section 2.1) and are meant to interact with the satellite through the satellite payload. *Payload service attackers* use a pre-defined service offered by the satellite, usually using a small antenna provided by the satellite's service provider, as shown in Figure 4b. Further, the traffic emitted by the attacker must be processed onboard the satellite, i.e., by parsing it, and is received on the Payload Data (PD) Fetcher interface. Additionally, we summarize attackers that have successfully compromised the payload, e.g., through the PDHS as *payload attackers*. They interact with the bus using the Link TC Fetcher interface that allows them to potentially escalate the attack from the payload to the bus.

4.3.3. Malicious Payload Service Hoster. These hosters hold the ability to host a custom service on the payload, i.e., by uploading *untrusted code*. The untrusted service of this user is executed on the UDHS. Consequently, the attacker has access to the UDHS to upload, update, or modify the service using the Untrusted Data (UD) Fetcher interface, as shown in Figure 4c.

4.3.4. Operators. Operators control the satellite's operations and exercise full control over the satellite as they issue commands over the TC Fetcher interface. While we do not consider attacks by *fully privileged operators*, we argue that *operators* are often divided into *fully privileged* and *semi-privileged operators*. This may apply to any sufficiently large group of operators where responsibilities and access privileges are separated. This scenario becomes more likely through the Satellite as a Service (SataaS) model, where access to a satellite is rented to untrusted third parties. In such scenarios, the untrusted parties might interact in a limited way, e.g., turn the payload on or off, prompting privilege escalation concerns.

Semi-Privileged Insider. A *semi-privileged position*, shown in Figure 4d, allows attackers to interact using non-critical TCs, e.g., to request telemetry or to manage non-vital payload systems. Hence, this attacker interacts with the satellite while confined, e.g., through ground-controlled upload restrictions. While this attacker's communication is accepted by the satellite's access control mechanism, the attacker does not have direct access to the cryptographic secrets.

5. Case Studies

We demonstrate the real-world applicability of our taxonomy and attacker models by analyzing three different satellites in detail. For each, we first conduct a technical analysis of the satellite's components, which we use to derive a satellite-specific threat model using our taxonomy as described in Section 3. Table 1 provides an overview of the analyzed satellites including key data. Then, we analyze the security of the satellites' firmware and Table 2 summarizes the main results. We find that each satellite is affected and successfully uncover multiple vulnerabilities. For two of the case studies, we experimentally verify the exploitability of our identified vulnerabilities and achieved arbitrary code execution on the CDHS, providing an attacker with full control over the satellite, which, to our knowledge, was never done before.

Satellite Analysis Challenges. During our analysis, we identified four key challenges that make the analysis diffi-

TABLE 1: Overview of the analyzed satellites and identified vulnerabilities.

Satellite	Orbit	Form	Launch	OBC	TCs	Strongest Attack Path
ESTCube-1	665 km	1U CubeSat	2013	ARM Cortex-M3	Unprotected	External Attacker → Seizure of Control
OPS-SAT	515 km	3U CubeSat	2019	AVR32 AT32UC3	Unprotected	External Attacker → Seizure of Control
Flying Laptop	600 km	60x70x90 cm	2017	Leon3 SPARC V8	Encrypted	Semi-Privileged Insider → TC Alteration

cult: (i) Satellites notoriously use a large variety of Instruction Set Architectures (ISAs) due to the versatile requirements of different satellites. In our survey (cf. Section 6), we found that 17 different satellites used eight different ISAs. This became an issue for *OPS-SAT*, as the AVR32 ISA barely has any support in analysis tools. (ii) Satellite firmware analysis is challenging due to the plethora of software components that are often custom-made for the satellites. Thus, documentation is typically lacking, and reverse engineering is time-consuming. (iii) Due to the large number of components connected to the CDHS, code paths can be hard to follow since potential interrupt sources from external devices can only be guessed. The RTOS design principle of data queues makes this even harder, as following a data flow is more challenging than following a well-referenced program flow. (iv) Space-domain specific protocols, such as the CCSDS family, are likely unfamiliar to most security analysts reversing satellite firmware, requiring more effort to study these protocols.

We conclude that the entry barrier to conducting security research is high, as several unique challenges exist.

Overview. At first glance, it might seem interesting to start with a case study on a large satellite in the GEO. However, the complexity of these large satellites makes them extremely challenging to study and hinders understanding security-relevant details. Hence, we first focus on the university-developed satellite *ESTCube-1*, for which we assume that the satellite's complexity is manageable.

As a second case study, we looked for a more complex satellite and found an ideal example in the *OPS-SAT*. Not only is the European Space Agency (ESA) involved in the development—which already contributes the knowledge of a major space agency—but the satellite is also an open research platform leading to an interesting attacker model.

Finally, we aimed for an even larger and more complex satellite. Here, *Flying Laptop* is a perfect example, as its FPGA-based computing setup works similarly to even much more complex satellites.

Analysis Scope. During our analysis, we focus on the OBSW in the CDHS, as attackers can use this module to gain full control of the satellite. In the OBSW, we focus on the TC/TM data channel, as it is the primary attack surface. Thus, we analyze incoming packet parsing, then follow the handling of the TCs before they are executed, and the actual execution of TCs. We also analyze the effects of notable TCs and searched for vulnerabilities in them. Note that all functionalities and vulnerabilities in the following case studies are in *active use* on the satellite unless we state otherwise.

Analysis Method. We used *IDA Pro* and *Ghidra* as tools during our analysis. Our initial analysis was manual and mainly involved reverse engineering of firmware binaries, where we analyzed the data flow from the COM systems to telecommand processing. We thereby manually reviewed and investigated the code for security issues. Additionally, we searched for references to functions prone to causing memory corruptions, such as `memcpy` and `strcat`. Finally, we used coverage-guided fuzzing through firmware re-hosting for *ESTCube-1* using *Fuzzware* by Scharnowski et al. [40].

Coordinated Disclosure. We responsibly disclosed our findings in a coordinated way to the respective satellite developers and *GomSpace*, a space SDK developer, while offering our help to solve problems. The *ESTCube-1* team already confirmed that they will fix the issues in the upcoming *ESTCube-2*. The teams of *OPS-SAT*, *Flying Laptop*, and *GomSpace* acknowledged that they received our reports but did not disclose further details or follow up on any of our subsequent requests.

Fixing Satellite Vulnerabilities. Estimating how long it takes to patch vulnerabilities is generally challenging, as it depends on the complexity of the underlying problem. Additionally, satellites face the unique challenge of having to upload a patched firmware. The *ESTCube-1* team told us that uploading a firmware image generally takes several days to a week, depending on the GS and link quality. This stems from the low bandwidth of UHF/VHF components (i.e., $9600\,\mathrm{bit/s}$) and shared bandwidths.

5.1. *ESTCube-1*

ESTCube-1 was Estonia's first satellite. It was developed by the *University of Tartau* in collaboration with the German Aerospace Center (DLR). The satellite was a 1U CubeSat in LEO and was decommissioned in 2015 while remaining in orbit. The primary purpose of the satellite was to demonstrate a novel propulsion method called the *electric solar sail* [42]. The secondary objective of the satellite was Earth observation on the visible spectrum. The satellite's second generation will launch in January 2023 and shares the majority of software components with *ESTCube-1*. Thus, only components specific to the new mission are developed

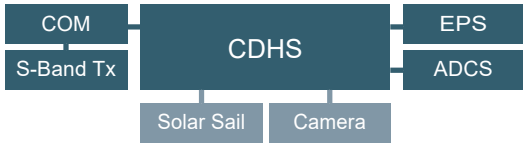


Figure 5: General overview of the *ESTCube-1* components

TABLE 2: Overview of vulnerabilities identified in the satellite firmware images. We only name the weakest (least-privileged) attacker model applicable. *Experimentally Tested* indicates whether we tested and confirmed this vulnerability.

Id	Satellite	Vulnerability	Attacker	Attack Surface	Outcome	Exploitable	Exp. Tested
1	*ESTCube-1*	Bypass Access Control	External Attacker	COM Rx	Arbitrary TCs	✓	✓
2	*ESTCube-1*	Dangerous TC	External Attacker	COM Rx	Arb. Code Exec.	✓	✓
3	*ESTCube-1*	Control Data Leak	External Attacker	COM Rx	Info Leak	✓	✓
4	*OPS-SAT*	Bypass Access Control	External Attacker	COM Rx	Arbitrary TCs	✓	✓
5	*OPS-SAT*	Dangerous TC	External Attacker	COM Rx	Arb. Code Exec.	✓	-
6	*OPS-SAT*	Critical TC Interfaces	Mal. Payload User	Link TC Fetcher	Arbitrary TCs	✓	-
7	*OPS-SAT*	TC Injection	Mal. Payload User	Link TC Fetcher	TC Injection	✓	-
8	*OPS-SAT*	Vulnerable Library	None	None	Arb. Code Exec	✗[a]	✓
9	*OPS-SAT*	Vulnerable TC	External Attacker	COM Rx	Arb. Code Exec	✓	✓
10	*Flying Laptop*	Missing TC Authentication	External Attacker	COM Rx	Arb. Code Exec	?[b]	-
11	*Flying Laptop*	Dangerous TC	Fully-Priv. Insider[c]	TC Fetcher	Arbitrary TCs	✓	-
12	*Flying Laptop*	Trusted Size Field	Semi-Priv. Insider	TC Fetcher	TC Alteration	✓	-
13	*Flying Laptop*	Inconsistent Size Field	Semi-Priv. Insider	TC Fetcher	TC Alteration	✓	-

[a] The vulnerable library function is not called in the analyzed satellite, but this library is also used by roughly 75 spacecraft and NASA [41].
[b] The encryption only ensures confidentiality but not integrity, allowing for ciphertext-only-based modifications.
[c] Only *fully-privileged* insiders could abuse this, which we do not consider (cf. Section 4.3.4). Alternatively, an external attacker bypassing the access protection may also exploit this.

from scratch, which implies that the following results are likely to impact *ESTCube-2* as well.

5.1.1. Technical Analysis. Figure 5 shows an overview of the *ESTCube-1*, which deploys two custom payload components [7]. The satellite's design is rather simple, as all components are directly controlled by the CDHS without a dedicated PDHS. The CDHS uses a redundant *STM32F103VF* ARM OBC with a *FreeRTOS* based OBSW.

The TC/TM traffic processed by the CDHS is received over the COM, which contains multiple antennas. One Ultra High Frequency (UHF) antenna is used for TC/TM traffic with the GS, and a different antenna listens on Very High Frequency (VHF) for an emergency reset signal. Additionally, there is an S-band antenna to downlink image data from the camera. Notably, the design of the COM includes no access protection or encryption mechanisms.

Internal Communication Protocol. *ESTCube-1* components use the custom Internal Communication Protocol (ICP) to communicate with the CDHS and the GS. The protocol does not use any security measures, such as encryption or authentication, and is designed to be straightforward. It uses a simple address scheme, where each component, including the GS, has an ID (e.g., GS $\hat{=}$ 5, and CDHS $\hat{=}$ 2). When parsing the packet in the CDHS, the ICP payload is used as TC packet and forwarded to the command scheduler, which eventually executes the command. Ultimately, the protocol presents a minimal solution to send ordered packets within a small mesh network of components.

5.1.2. Threat Model. Figure 6 displays the threat model derived from our taxonomy (cf. Section 3), including the vulnerabilities that we identified. Since the satellite's payload does not offer a PDHS or external receivers for structured data, the threat model only includes the bus. The S-Band antenna is not included as it is only used for transmission. *ESTCube-1*'s attack surface is defined through the interfaces. Hence, only *external attackers* with a *custom GS* against the *COM Rx* and *semi-privileged operators* connected to the *TC Fetcher* are relevant (cf. Section 4.3).

Experimental Setup. We rebuilt (parts of) the satellite in our lab to test our results experimentally. We recreated the satellite's CDHS hardware using the same STM32 microcontroller on a breakout board with a J-Link debugger probe. Further, we connected the satellite to actuators representing the payload control functionalities. This setup runs the OBSW unmodified on the same hardware. Additionally, we connected a speaker to the port usually occupied by the *Solar Sail* (cf. Section 5.1.1). Building an exploit playing sound proves that we can control this (or any) port.

5.1.3. Security Analysis. We summarize our main findings.

Unsecured Telecommand Access. The most striking issue of *ESTCube-1* is the missing TC encryption and authentication, which results in a trivial *access control bypass* (cf. Section 3) on the COM. During the active commission of the satellite, *external attackers* with a custom GS could have issued arbitrary commands to the satellite.

Unfortunately, there seems to be no trivial fix, as the employed ICP protocol would have to be extended to allow for cryptographically secured interactions. This reveals the problem of using custom protocols for security-critical applications. Even when assuming that access protection is

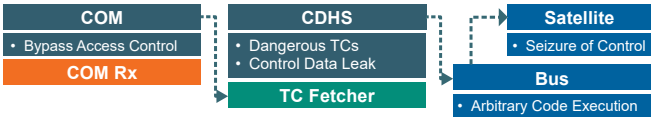


Figure 6: *ESTCube-1* threat model and vulnerabilities

in place, it would be a single point of failure due to the subsequently discussed *dangerous TC*.

Insecure-by-design TCs. Even with no access protection, a satellite should be designed so that TCs do not compromise the satellite's stability without further validation, as outlined in Section 3. Here, two specific TCs allow arbitrary reading and writing of memory. On the technical level, the attacker controls all parameters passed to `memcpy` through command arguments, such that these two TCs are *dangerous TCs*. Anyone with a custom GS could utilize them to gain remote code execution and seize control of the satellite.

Noteworthy, the ability to execute arbitrary code would allow an attacker to write firmware updates to the flash memory persistently, making the takeover irreversible. Other than modern operating systems such as Linux or Windows, which deploy defenses to prevent trivial exploitation of such vulnerabilities, the RTOS in *ESTCube-1* does not feature any such protections. In particular, neither ASLR nor stack cookies are used. To prove the impact of this vulnerability, we build an exploit, send our payload over the COM interface of our rebuilt satellite in the lab, and execute arbitrary code (in our case, we play sound over the connected speaker).

Trusted ICP Size Field. Upon receiving an ICP packet, the packet is passed through a *FreeRTOS* data queue to the command scheduler, which executes the associated command using the included arguments. We observed that a function parsing the command structure does not validate the "*length of arguments*" field against the total length of the ICP packet or payload. Thus, any *external attacker* can specify a malicious length field, which indicates that the arguments would be longer than they actually are. This causes a command handler function to use more bytes from the heap memory than intended, leading to a buffer overread. Hence, an attacker can include other data in the attacker-TC, which leads to a *control data leak* (cf. Section 3.1.2). Again, we verify that this works on the real satellite by testing it on our recreated hardware and manage to successfully exploit this vulnerability. The leak itself is reliable and is not impacted by environmental conditions, but extracting specific secrets depends on the heap layout. Noteworthy, this vulnerability is similar to the well-known *OpenSSL Heartbleed* vulnerability [43].

5.2. *OPS-SAT*

OPS-SAT is the first CubeSat directly operated by ESA. The satellite was developed by the *Graz University of Technology*, launched in 2019 into LEO, and remains in active use. The satellite offers a versatile platform to run scientific experiments and technology demonstrations. Notably, people independent of ESA and without specific satellite knowledge can develop experiments through a dedicated open-source framework [44], [45]. Hence, the satellite provides a rare scenario with untrusted third parties performing experiments, which may become more widespread in the future as SataaS becomes more prevailing [46], [47]. Thus, *OPS-SAT* is an excellent case study to research this threat scenario early on. Curiously, *OPS-SAT* was subject to a recent security competition. There, the *satellite bus* was considered off-limits, likely due to the criticality of potential exploitation, underlining the impact of our findings [48].

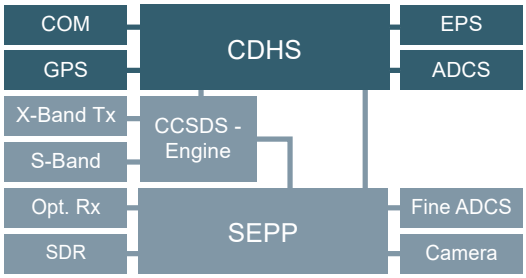


Figure 7: Overview of the *OPS-SAT* components

5.2.1. Technical Analysis. *OPS-SAT* is split into a satellite bus and payload according to our reference satellite model (cf. Section 2.2), where the payload deploys the experiments and related devices. Figure 7 highlights the most notable parts and a simplified version of their connections.

Satellite Payload. The satellite payload in the lower part of Figure 7 is centered around the cold-redundant Satellite Experimental Processing Platform (SEPP), which provides a dual-core ARM Cortex A9 processor. Further, the SEPP is connected to a Software-Defined Radio (SDR), a camera, and fine adjustable ADCS, which is required for the optical receiver (*Opt. Rx*). Additionally, the payload deploys an S-Band radio transceiver for high-bandwidth data transmits and an X-band transmitter for higher bandwidth downlinks. The traffic from these radios is handled through the CCSDS engine, which implements CCSDS hardware decoding in a Field-Programmable Gate Array (FPGA) connected to the OBC and the SEPP.

Satellite Bus. The satellite bus in the upper part of Figure 7 has a common satellite bus design and is centered around redundant *NanoMind A3200* OBCs [49]. The COM deploys a UHF/VHF radio to communicate with the GS.

The OBSW uses the *FreeRTOS* v8.2.1 RTOS and is built using the *GomSpace NanoMind SDK*, which provides hardware abstractions and common functionalities, such as file system access and a parameter database. To process TCs from the COM, the Cubesat Space Protocol (CSP)—implemented in the open-source library *libCSP*—is used. The protocol is commonly used for small satellites due to the minimal network protocol. The CSP is thereby used to transmit the SPP, which in turn contains the telecommand data. Additionally, the OBSW can also receive SPP-wrapped TCs from the CCSDS engine and the SEPP.

5.2.2. Threat Model. Figure 8 presents the threat model for *OPS-SAT* based on our threat taxonomy (cf. Section 3). The CDHS features two *TC Fetchers*, one over an *I2C* bus connected to the UHF/VHF COM and one connected over a Controller Area Network (CAN) bus to the S-Band COM, which features a CCSDS engine represented by the *S-Band COM* box. The *CAN - TC Fetcher* also receives the TCs from the *payload*.

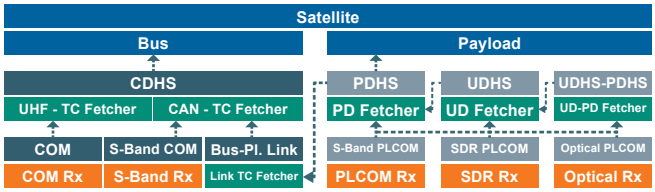


Figure 8: The *OPS-SAT* threat model

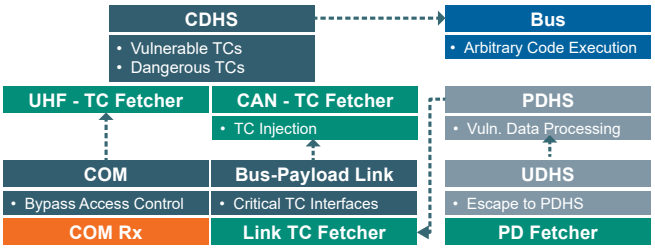


Figure 9: An overview of the vulnerabilities identified in the satellite bus and their attacker paths

The *payload* features a PDHS, which is called SEPP on *OPS-SAT*. This PDHS deploys an UDHS to run untrusted scientific experiment applications in an isolated environment. As described in Section 3.1.2, the environment running the scientific application is the UDHS, whilst the application itself is modeled as nested PDHS (the UDHS-PDHS). Finally, all components in the payload have access to the *S-Band PLCOM*, which is physically the same CCSDS-engine connected radio as the *S-Band COM*, and they have additional access to a SDR (*SDR PLCOM*) and the optical receiver *Opt. PLCOM*.

Hence, *OPS-SAT* exposes a considerable attack surface through the abundance of communication options. All attackers described in Section 4.3 are relevant to the security considerations of *OPS-SAT* and use the interfaces displayed in Figure 4, while the *UD-PD Fetcher* is a *PD Fetcher*.

5.2.3. Experimental Setup. Since the *NanoMind* board used on the real satellite is costly and similar boards are out of production, we resorted to software-based emulation and static analysis. To our knowledge, there is no comprehensive emulator for this ISA. Hence, we implemented the AVR32UC3 ISA in QEMU from scratch to provide us with an accurate testing and evaluation environment [1]. We omitted the emulation of peripherals that physically interact with the real world, while implementing a minimal-working version of peripherals involved in the execution of the CDHS, such as flash memory.

5.2.4. Security Analysis. In the following, we briefly summarize our main findings. Figure 9 presents the vulnerabilities that we identified in the bus. In Figure 9, we omit the blue satellite box containing the *seizure of control* capability and the *PD Fetcher* interface due to space constraints

Bypass Access Control. *OPS-SAT* uses the S-Band radio coupled with the CCSDS engine, which, to our knowledge, decrypts S-band traffic. The COM radio is directly connected to the OBC, where the OBSW uses the CSP protocol, which offers eXtended Tiny Encryption Algorithm (XTEA) encryption and Hash-Based Message Authentication Codes (HMAC) authentication. However, these capabilities are disabled via a compile-time flag. Hence, the COM does not offer any protection, and *external attackers* can command the satellite by executing arbitrary TCs, which includes anyone with a custom GS.

1. The source code of our implementation can be found at https://github.com/CISPA-SysSec/SpaceOdyssey-QEMU-AVR32

Another interesting observation is that the COM handles a larger range of commands than the S-Band antenna, as it has access to additional command services. The additional services are the *libCSP* default handlers (i.e., ping replies, uptime checks, and reboots), the parameter service for flight parameter editing, the ADCS server, and a sensor telemetry service. Hence, the unprotected COM exposes more attack surface than the protected S-band COM.

Unsecured Software Updates. *OPS-SAT* uses a flash file system to store files, including the firmware image. Existing TCs allow to create new files and write to them, providing the capability to upload a malicious firmware image onto the satellite. To change the filesystem path pointing to the current image, *critical commands* must be enabled, which is a global Boolean flag in the satellite's settings. Crucially, changing this flag can be done via a TC that does *not* require additional verification. Hence, *external attackers* can conduct arbitrary firmware updates, which allows them to seize control over the satellite. Interestingly, similar *critical* functionalities are hidden behind the same flag, indicating that engineers were aware of its critical importance but decided not to implement further protection.

Critical TC Interfaces. Both the SEPP and the CCSDS engine (cf. Figure 7), which in Figure 8 are the PDHS and the *S-Band COM* + *S-Band PLCOM*, are connected via the same CAN bus to the CDHS. All packets in this CAN bus are fetched using the same code in *CAN - TC Fetcher*, which does not differentiate between the origin of these packets. Hence, TC packets coming from the protected *S-Band COM* and from the PDHS are processed the same way, exposing all TCs offered over the *S-Band COM* also to the PDHS. This includes the *unsecure software update* described before, which leads to a *critical TC interface* vulnerability. Hence, any attacker with control over the PDHS can issue critical TCs to the CDHS and conduct a malicious firmware update. Theoretically, attackers are not supposed to have control over the PDHS, which implements isolation to the UDHS, but this isolation can be broken as shown by Didelot [50]. We discuss this further in Section 5.2.5.

Trusted External Input. It is crucial that applications do not trust external input, especially regarding buffer sizes. Nevertheless, the CAN bus implementation accepts packets from the S-Band antenna and the SEPP of arbitrary size. While individual CAN bus packets are limited to eight bytes per transmission, an arbitrary number of packets can

be sent before the *transmission-end* marker is sent. Meanwhile, these packets are copied into a *static* buffer, allowing an attacker to write beyond the intended bounds.

As seen in Figure 9, this vulnerability poses a *data injection* threat to the *CAN - TC Fetcher*, which receives telecommands from the PDHS (PDHS $\hat{=}$ SEPP). However, the vulnerability can not be exploited from the *S-Band COM*, as it consists only of an FPGA with insufficient fine-granular control over the data transmitted on the CAN-bus. Contrary, the PDHS has full control over the transmitted data, allowing attackers that compromised the PDHS to inject data into the *Link-TC Fetcher*.

The vulnerability does not yield an attacker new capabilities due to the existence of beforementioned *critical TC interfaces*, which expose the critical TCs that enable malicious firmware updates to the PDHS.

Vulnerable Libraries. The *GomSpace NanoMind SDK* in the OBSW utilizes the *uffs* library, which implements a low-cost flash file system [33]. Interestingly, the library is used on roughly 75 spacecrafts (cf. Section 7.1) and, according to the library's author, used by NASA [41]. We identified a stack-based buffer overflow vulnerability in the file renaming procedure, where the name of the new file is copied into a buffer of static size without any size check, resulting in arbitrary code execution.

We experimentally verified that this vulnerability can be exploited to gain arbitrary code execution. In *OPS-SAT* this function is only exposed to an inaccessible Universal Asynchronous Receiver-Transmitter (UART) debug-port, posing no security threat to *OPS-SAT* in its current state. Still, moving files is a reasonable file system interaction to be exposed via TC to *semi-privileged attackers*. Hence, if one of the other roughly 75 spacecrafts implements such functionality, it is likely vulnerable.

Memory Corruption in TC. We identified a buffer-overflow vulnerability in the firmware, more precisely in the ADCS server's functionality that allows specifying a log file. While the log file name is stored in a static buffer, the `strcat` function that writes into this buffer copies the string from a TC that can be larger than the static buffer, resulting in a *vulnerable TC*. Hence, this vulnerability yields an attacker control over the satellite through arbitrary code execution in a single TC via the *UHF - TC Fetcher* interface, as opposed to the malicious firmware update, which requires multiple TCs. We successfully exploited this vulnerability in our AVR32-QEMU emulation.

5.2.5. Payload Vulnerabilities. Complementary to our research, Didelot conducted a non-academic security analysis of the *OPS-SAT* payload [50]. Since *OPS-SAT* features a rare chance to study a complex payload architecture with an UDHS, this provides an excellent opportunity to show our taxonomy's broad applicability. Hence, in the following, we summarize the payload exploitation present in Figure 9.

The PDHS deploys a Yocto Linux, which contains the UDHS. Applications for the UDHS are developed in Java using a custom framework from ESA. Then, on the satellite, the Java applications are running in an isolated fashion and

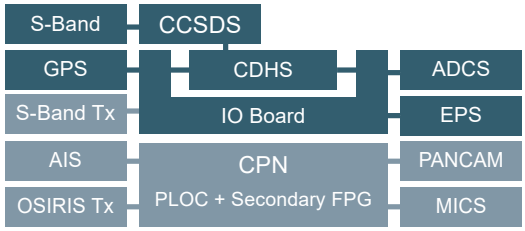


Figure 10: Overview of the *Flying Laptop* components

are only supposed to interact with the PDHS through a *Supervisor*, to which the interface is implemented in Java. Didelot identified three notable vulnerabilities. The first is an *escape to PDHS* vulnerability, where a command injection into a supervisor's shell command with input from the UDHS application grants the attacker remote code execution on the PDHS. The second vulnerability concerns the PDHS, where an application deploys the uploaded experiments provided in the *Zip* format. By defining a target path using path traversal outside the intended unpacking directory, attackers can deploy arbitrary files on the Linux filesystem, resulting in a *vulnerable data processing* vulnerability. While attackers have escaped the UDHS isolation, they do not have *root* privileges yet. However, another *vulnerable data processing* vulnerability generates a list of *sudo*-capable users from a directory with non-root access. Hence, any attacker with access to the filesystem can gain *root* privileges. ESA has fixed these vulnerabilities following a responsible disclosure process by Didelot [50].

5.3. *Flying Laptop*

Flying Laptop is a small satellite launched in 2017, operated by the *Institute of Space Systems* at the *University of Stuttgart* and developed in cooperation with *Airbus Defense and Space* [51]. It is still in active use today. The satellite serves as a technology demonstrator for a *future low-cost platform* and includes an OBC from *Airbus*, which provides valuable insights on how global defense and space companies develop satellites. Additionally, the complexity and size of the satellite, classified as *medium/large*, makes it an excellent case study leading up to more complex satellites.

5.3.1. Technical Analysis. Figure 10 presents a technical overview of *Flying Laptop* with the upper part representing the bus and the lower part being the satellite payload.

Satellite Bus. The bus follows the common satellite architecture (cf. Section 2.2) and is centered around a redundant *Leon3 SPARC* microprocessor as the OBC. The OBSW uses the RTEMS RTOS *v4.10.2* for TC/TM traffic communication over the COM. The COM is connected through a *SpaceWire* interface, which is a bus akin to a CAN bus designed by ESA for space applications with tight integration for the CCSDS protocol. Traffic to and from the *SpaceWire* port is handled by a CCSDS de/encoding board coupled with an S-band antenna. Curiously, the decoding is performed transparently, which means that the OBSW still

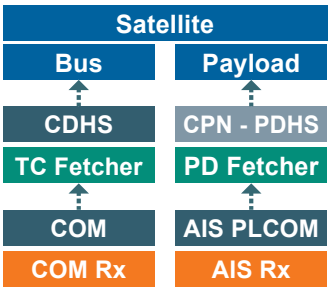


Figure 11: The *Flying Laptop* threat model

receives a CCSDS packet. Hence, the OBSW implements both a CCSDS and an SPP parser to receive TCs.

Satellite Payload. The purpose of the satellite payload is to handle the data from the two cameras and the Automatic Identification System (AIS) for maritime vessel tracking. Therefore, it is centered around the *Central-Processing Node (CPN)* that uses a full FPGA-based computing setup on the *Payload OBC (PLOC)* with volatile and persistent memory. After processing the data, it is sent to the ground using the S-band transmitter or the optical transmitter system *OSIRIRS*. Finally, the CPN deploys a secondary FPGA that initializes the PLOC-FPGA, allowing for post-launch firmware updates.

5.3.2. Threat Model. Figure 11 features the threat model of *Flying Laptop* derived from our taxonomy. Surprisingly, we did not identify a *Bus-Payload Link*, meaning the bus and payload are fully isolated. On the *payload*, we only identified the AIS receivers as an attack surface, the data of which is processed on the PDHS.

5.3.3. Security Analysis. The main results of our security assessment are summarized below. We could not verify our findings beyond static code analysis, as building a proper emulation was challenging due to a lack of peripheral documentation, and we had no access to a hardware model. Figure 12 summarizes the vulnerabilities that we found during our analysis.

Missing TC Authentication. To our understanding, the CCSDS engine in the COM implements the decryption of TCs. Crucially, it does not implement authentication of TCs, which is only planned for the follow-up satellite [52]. Hence, *external attackers* may use replay attacks or forged-ciphertext attacks to bypass decryption, depending on the supported encryption type. Currently, this poses a comparably limited security risk, considering that the other satellites investigated do not even implement encryption. Regardless, this shows that either the confidentiality of TCs was a concern, which we consider unlikely, or that there was the erroneous assumption that encryption would protect TC integrity.

Dangerous TC. *Flying Laptop* offers a `memcpy` TC similar to what *ESTCube-1* does, where all arguments of a `memcpy` call can be supplied through a telecommand, exposing a *dangerous TC*. Attackers that bypassed the access control can use this to execute arbitrary code. To exploit

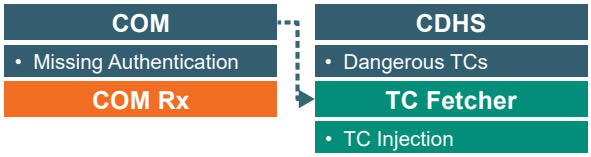


Figure 12: Overview of *Flying Laptop*'s vulnerabilities

this, attackers need unrestricted access to the *TC Fetcher* interface, which only *fully-privileged operators* have due to the *access protection* in place. We do not consider this threat scenario in this work but stress that this breaks one link in the security chain, making access protection a single point of failure.

Trusted Size Field. We found that the SPP implementation in the CDHS fully trusts the packet's size field, which is never sanitized and leads to a buffer overflow vulnerability. *Flying Laptop* fully trust the size field contained in SPP even after validating the CCSDS size field. By doing this, attackers can include data from a previous TC in their attacker command leading to *TC Alteration*, which *semi-privileged operators* can exploit via the *TC Fetcher* to effectively execute different commands, but only TCs that the *operator* is authorized for on the GS (cf. Section 4.3.4).

Inconsistent Size Field. When working with a buffer, the associated length field must be updated whenever the buffer is changed. However, we found a padding mechanism in the CCSDS parser that skips leading padding by incrementing the buffer pointer without updating the corresponding size field. An attacker can use this to force the CCSDS parser to read other TC packet data by overreading the original buffer, effectively including other TC data in the attacker's TC, allowing *semi-privileged operators* to alter their TC after upload. The vulnerability is also exploitable via the *TC Fetcher* interface like the *trusted size field*.

6. Satellite Community Survey Results

To learn more about the technical context of modern satellites and to better understand the awareness of industrial practitioners regarding potential security risks, we surveyed professionals that develop satellites. We use the survey results [2] to embed the results of our case studies in a larger context, allowing us to draw broader conclusions about satellite firmware security.

6.1. Survey Background

In the following, we outline our survey design, discuss ethical considerations, and provide an overview of our participant selection process.

6.1.1. Survey Structure. We divide the survey into five sections, asking about (i) demographics, (ii) background information on the participant, (iii) details of a satellite they worked on, (iv) personal experience with security of

2. A graphical representation of the questions and the results is available at https://github.com/CISPA-SysSec/SpaceOdyssey-Survey

satellites, and (v) security-specific details of a satellite. For demographics, we asked for age, gender, and country of residence. Then we asked the participants how they are associated with satellite development and how many satellites they had worked on. If they had worked on more than one, we asked them to focus on one specific satellite for the satellite-specific questions. In the following sections, we refer to the surveyed satellite as "*their satellite*" in survey-specific passages. Following that, we asked respondents for their personal thoughts on security aspects of satellites, which we use to formulate our attacker model (cf. Section 4). Finally, we asked several technical questions about the security aspects of the surveyed satellite, while also giving participants the option to abstain on individual questions. After the last questions, we offered the interviewees to answer the satellite-specific questions again for a different satellite.

6.1.2. Ethical Considerations. We developed our survey in collaboration with a professional survey development team, and the survey was approved by our internal review board. When designing the survey, we also kept best practices like the framework outlined by the Menlo Report [53] in mind. The survey includes an informed consent section that informs the respondents about the voluntary participation, the survey's purpose, a description of the procedure, the option to skip questions, the collected data, our intent to publish the results, and a privacy notice and contact information.

Due to the sensitive nature of satellite security, we provided a fully anonymous environment by not tracking any personally identifiable information, such as IP addresses, geographic locations, or personalized links. Further, we allowed participants to skip questions, with the response option *"Prefer not to say / Don't know"* to corresponding questions, which we refer to below as *abstains*.

6.1.3. Participants and Survey Access. We conducted the survey using a self-hosted instance of *Qualtrics*. We distributed the survey via a non-personalized link and a non-personalized QR code. We did not publish access to the survey, such as through social media, but distributed it directly to eligible teams and individuals that develop satellites. We took care, to the best of our abilities, that only people with the required knowledge and expertise had access to the survey. Thus, we deliberately did *not* address engineers developing a specific hardware component only and are otherwise not integrated into the satellite's development life cycle. We thereby addressed eligibles in companies, academic institutions, and government as well as international institutions.

In total, we received 19 survey responses covering 17 satellites and the participants have worked on a total of 132 satellites. Three participants answered questions about their personal experiences but did not answer questions about individual satellites, whilst one participant answered questions about two satellites. Further, the sum of satellites that the participants stated that they have worked on during their career is 132. It should be noted that even with the 19 valid responses we received, it took about four months to convince people to complete the survey. In general, we observed that people were very reluctant to share any details about their satellites and their security aspects.

6.2. Key Results

Looking back to our case studies, we consider the absence of proper TC access protection the most severe issue, as it leads to *bypass access protection* vulnerabilities in *(PL)COM* components (cf. Section 3.1.2). Additionally, we uncovered several memory corruption-based vulnerabilities in CDHS components, which is surprising given one would assume that these critical space systems undergo rigorous testing. Hence, we formulate three questions:

1) How common are unprotected TC interfaces?
2) Where are standardized space protocols used?
3) How are software components security-tested?

6.2.1. Missing Telecommand Protection. Our case studies found that both *OPS-SAT* and *ESTCube-1* expose *bypass access protection* vulnerabilities in COM components, raising the question of how common this issue is. Unfortunately, there is no standard way to test this, e.g., by pinging satellites and checking for a response indicating successful TC processing. While there is research on access protection [54], [55], to the best of our knowledge, no research has evaluated the prevalence of such protections.

In our survey of 19 professionals, we received responses for 17 satellites: For nine satellites, it was stated that measures to prevent third parties from controlling the satellite were implemented, while for three satellites, participants stated outright that there are *no* measures. The other five satellite responses declined to comment or did not know for sure. While only 53% respondents are confident to state that there are defenses (which is already quite low), the defense mechanisms used paint a grim picture. To prevent third parties from controlling the satellite, protocol obscurity (5 votes) is tied with protocol encryption (5 votes). Further, all 5 votes that stated "*protocol obscurity*" also stated that they have measures in place to prevent third-party access, which shows the prevalence of *security by obscurity* (cf. Section 4.1). In addition, only 5 respondents stated that they use access control on the satellite, indicating that TC authentication is used less frequently than encryption.

In summary, according to our survey and experimental findings, most small non-constellation LEO satellites have no TC protection to defend against malicious access and hence expose *access protection bypass* vulnerabilities. While our results show that *protocol obscurity* is still prevalent, they also point to a deeper issue with satellite on-board security: It appears that even basic remote access protection has not yet arrived in the architectural planning phase in many of these satellites.

6.2.2. Standard Security Protocol Usage. During our case studies, we found that *ESTCube-1* uses a custom, minimal protocol, *OPS-SAT* uses a CSP/SPP combination as well as

TABLE 3: Overview of the number of satellites that use *standardized* and *custom protocols* per satellite weight.

	<1 kg	1–50 kg	50–100 kg	>100 kg
Standard Protocol	0	1	1	4
Custom Protocol	0	6	1	0
Abstains	0	3	0	1
$\sum$	0	10	2	5

a full CCSDS stack, and *Flying Laptop* uses a full CCSDS stack for TCs in the COM. This is in line with our discussion that more complex satellites use more complex solutions (cf. Section 5).

Interestingly, *ESTCube-1* uses a custom ICP protocol (cf. Section 5.1.1), while *Flying Laptop*'s CCSDS stack specification is readily available. This appears counter-intuitive, as one would assume that small development teams would not want to invest resources to create a fully custom communication protocol when they could use an existing solution. To further test this assumption, we use our survey in which 7 participants stated that they do *not* use a standardized protocol (which is the majority), while only 6 indicated that they do. Table 3 shows the relationship between satellite weights and the usage of a custom protocol. While satellite weight and complexity are technically not the same, we think they correlate as increasingly heavy satellites also require higher launch costs and potentially more expensive components, naturally increasing the project's complexity. Hence, from the table and our case studies, we can derive that more complex satellites are more likely to use standard protocols, raising the question of why this is the case.

6.2.3. Lack of Security Testing. Security testing methods such as fuzzing, symbolic execution, (bounded) model checking, and penetration testing could have identified the vulnerabilities uncovered during our case studies in CDHS components. However, only 2 participants stated that they used penetration testing, and 1 participant stated that (bounded) model checking was used. None of the interviewees used fuzzing or symbolic execution, indicating a need for improvement. The vast majority of people surveyed stated that they use other testing methods: unit-testing (14 votes) and hardware/software/model in-loop testing (14/10/6 votes). Ultimately, these techniques aim to ensure the correctness of software, which makes sense as a developer's main concern for space systems. As such, the lack of security testing is not rooted in a lack of testing but rather in the unawareness of advanced security-focused testing methods.

7. Discussion

We now reflect on the security issues we identified and discuss potential limitations of our work.

7.1. Vulnerable Software Components

During our case studies, we encountered several memory corruption vulnerabilities, e.g., for *OPS-SAT*, we found buffer overflows in the *GomSpace SDK* and the OBSW. Similarly, we encountered trusted SPP length fields in *Flying Laptop*, showing how vulnerable space domain software and widespread libraries are.

While the vulnerability in the *GomSpace SDK* itself can be fixed, the source of the problem lies in the *uffs* library, which received the last code update in 2014. Hence, it is unlikely that the library itself will be fixed and that fixes will be upstreamed. Exact usage numbers of the library are not published. According to a report, the *GomSpace NanoMind* (and thus the SDK as well as the *uffs* library) is part of more than 75 space missions as of 2022 [56]. In addition, the developers of *uffs* stated that their library is used by NASA [41]. Hence, our findings directly impact a sizeable number of spacecraft. While this direct impact is significant, it also points to a deeper issue concerning the type of vulnerabilities. Ultimately, these well-known types of vulnerabilities should not occur in critical modern space systems.

Another library encountered during our analysis is *libCSP*, which has similarly concerning vulnerabilities. According to the GitHub page, the library's implementation of cryptographic primitives suffers from predictable numbers only used once (nonce) [57], timing side channels on MAC verification, and replay attacks [58]. These issues likely impact a sizeable number of spacecraft, as the library is rather popular with 196 forks and 345 stars [59]. The vulnerabilities indicate that security was of concern, but the necessary knowledge on securely implementing cryptographic primitives was inadequate.

Finally, straightforward exploitation of the vulnerabilities presented here could be prevented by modern operating systems that implement measures such as Address Space Layout Randomization (ASLR), non-executable stacks, and separated (*root*-) privileges. However, the RTOS systems that we encountered did not implement any of these methods on the satellites. Our work is a call to action to finally implement modern defenses on satellite operating systems to prevent straightforward exploitation.

In conclusion, the simple nature of these vulnerabilities is a main reason for concern and shows that little security research from the last decade has reached the space domain.

7.2. Challenging Protocol Standards

In Section 6.2.2, we uncovered that, counter-intuitively, more complex satellites are more inclined to use standard protocols, while small ones rather develop custom protocols.

To our understanding, the go-to choice for standard protocols is the CCSDS family of protocols (cf. Section 2.3). However, as previously stated, this ecosystem is complex with at least 14 different protocols and options to use *Internet protocols* (i.e., IPsec or TCP) [19]. Hence, selecting the protocols for a use case is non-trivial, as each selected

protocol requires an individual investigation of the potential issues and advantages in complex standard documents. While the same could be said for many domains, the space domain suffers particularly from an absence of best practices, likely due to the *security by obscurity* principle (cf. Section 4.1). Similarly, existing implementations are rare, making it hard to find role models. In particular, we found a Java-based implementation [60] that is unsuitable for the CDHS systems we encountered.

Finally, putting aside industry limitations in using CCSDS, we could not find a single mention of CCSDS in top-tier security research conferences in the past ten years. This is rather astonishing for a protocol that has been under development since 1995 and is thus as old as SSL 2.0.

7.3. Limitations and Generality

In this paper, we faced several constraints concerning our case studies and survey, as described below.

Case Studies. Due to scarce access to firmware images, we are limited to a small number of satellites to study. However, even with a larger number of study targets, manual analysis is tedious due to a lack of documentation. Regardless, we believe that we have been able to select and analyze a representative sample of satellites that allows us to present unique space domain challenges.

Survey. Our survey covers only a comparatively small number of 19 satellite professionals. However, we found that the community is small and secretive. Also, the number of satellites the participants of our survey have worked on adds up to a non-negligible number of 132, which is significant in itself, but even more so when considering that we conducted our survey only in Europe. Hence, we believe that our sample is sufficient for our consideration.

8. Related Work

Satellite security research has only recently gained traction in light of the *New Space Era*.

Satellite-Based Network Security. Pavur et al. have shown a lack of security in DVB-S and VSAT networks [61], [62]. Additionally, Giuliari et al. investigated the resilience of LEO constellations against DoS attacks [63]. In light of this research, there have also been attempts to propose mitigations. Pavur et al. proposed a *QUIC*-based VSAT encryption scheme [64], Jedermann et al. proposed an orbital authentication scheme [25], and Oligeri et al. proposed spoofing detection methods for Iridium [65].

However, these topics solely focus on the *payload* network security aspect, where the satellite is merely acting as a *bent pipe*, while the internals are of no special relevance.

Satellite Security. Falco [13], as well as Livingstone and Lewis [66], have discussed key aspects of why space security differs from terrestrial security. Falco et al. also introduced a framework to analyze the threats against CubeSats [16]. Further, Manulis et al. have discussed aspects of the *New Space Era* that must be considered in future security research [67]. Additionally, two reports on space asset threats by Harrison et al. have pointed out the dangers posed by attacks [68], [69]. Ultimately, there is a plethora of additional research on this topic, albeit of theoretic nature [70]–[73] or involving simulations [23].

Hence, to our knowledge, our work is the first to provide real-world insights into the security of live satellites and the ecosystem evolving around them. As previously pointed out by Pavur et al., the industry, for a long time, acted as a gatekeeper to satellite security research by not providing any software for research [15]. In this work, we finally overcome this challenge and provide an impactful security analysis.

9. Conclusion

In this paper, we explore the state of satellite security and systematically analyze the attack surface. We formulate a generally applicable taxonomy of threats against satellite firmware that allows us to derive satellite-specific threat models, thereby overcoming prevailing but outdated assumptions. We then study three satellites and find that all expose different types of software vulnerabilities and largely insufficient protections against attackers. Based on our taxonomy, we show that two satellites expose vulnerabilities that enable attackers to execute arbitrary code, allowing them to seize control of the satellite using firmware vulnerabilities, which has not been shown before. Challenging our results with a survey amongst satellite professionals, we confirm our findings and provide valuable insights into the unique challenges of satellite security. We hope that this work will serve as a starting point to study the security and privacy aspects of satellites in the New Space Era.

Acknowledgments

We thank Annabelle Walle and Michael Schilling of the empirical research support group and Avian Krämer of the CISPA Helmholtz Center for Information Security for their assistance. This work was funded by the European Research Council (ERC) under the consolidator grant RS^3 (101045669) and by the German Federal Ministry of Education and Research (BMBF, project CPSec – 16KIS1564K). The work was partially supported by the MKW-NRW research training group SecHuman.

References

[1] O. Kodheli, E. Lagunas, N. Maturo, S. K. Sharma, B. Shankar, J. F. M. Montoya, J. C. M. Duncan, D. Spano, S. Chatzinotas, S. Kisseleff *et al.*, "Satellite Communications in the New Space Era: A Survey and Future Challenges," *IEEE Communications Surveys & Tutorials*, 2020.

[2] United Nations Office for Outer Space Affairs (UNOOSA). (2022) Online Index of Objects Launched into Outer Space. [Online]. Available: https://www.unoosa.org/oosa/osoindex/

[3] E. Kulu, "Satellite Constellations-2021 Industry Survey and Trends," in *Small Satellite Conference*, 2021.

[4] ——. (2019) Nanosats Database. [Online]. Available: https://www.nanosats.eu/

[5] A. Camps, "Nanosatellites and Applications to Commercial and Scientific Missions," *Satell. Mission. Technol. Geosci*, 2020.

[6] S. Tsitas and J. Kingston, "6U CubeSat Commercial Applications," *The Aeronautical Journal*, 2012.

[7] I. Sünter, A. Slavinskis, U. Kvell, A. Vahter, H. Kuuste, M. Noorma, J. Kutt, R. Vendt, K. Tarbe, M. Pajusalu *et al.*, "Firmware Updating Systems for Nanosatellites," *IEEE Aerospace and Electronic Systems Magazine*, 2016.

[8] R. L. Staehle, B. Anderson, B. Betts, D. Blaney, C. Chow, L. Friedman, H. Hemmati, D. Jones, A. Klesh, P. Liewer *et al.*, "Interplanetary CubeSats: Opening the Solar System to a Broad Community at Lower Cost," *NTRS - NASA Technical Reports Server*, 2012.

[9] Amazon Web Services. (2022) AWS Ground Station. [Online]. Available: https://aws.amazon.com/ground-station

[10] Microsoft Azure. (2022) Azure Orbital - Satellite Ground Station and Scheduling Services for Fast Downlinking of Data. [Online]. Available: https://azure.microsoft.com/en-us/services/orbital

[11] M. Chlosta, D. Rupprecht, T. Holz, and C. Pöpper, "LTE Security Disabled: Misconfiguration in Commercial Networks," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[12] Committee on Science, Space, and Technology, "ASA Cybersecurity: An Examination of the Agency's Information Security," *Hearing before the Subcommittee on Investigations and Oversight, Committee on Science and Technology House of Representatives*, 2012.

[13] G. Falco, "The Vacuum of Space Cyber Security," in *AIAA SPACE and Astronautics Forum and Exposition*. American Institute of Aeronautics and Astronautics, 2018.

[14] B. Driessen, R. Hund, C. Willems, C. Paar, and T. Holz, "Don't Trust Satellite Phones: A Security Analysis of Two Satphone Standards," in *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[15] J. Pavur and I. Martinovic, "Building a Launchpad for Satellite Cybersecurity Research: Lessons from 60 Years of Spaceflight," *Journal of Cybersecurity*, 2022.

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

[17] M. Manulis, C. P. Bridges, R. Harrison, V. Sekar, and A. Davis, "Cyber Security in New Space," *International Journal of Information Security*, 2021.

[18] S. Cooper, "CCSDS Mission Operations Services in Space," in *SpaceOps*. Citeseer, 2012.

[19] Consultative Committee for Space Data System, "Overview of Space Communication Protocols," Consultative Committee for Space Data System, Standard, 2014.

[20] I. A. Sanchez, G. Moury, and H. Weiss, "The CCSDS Space Data Link Security Protocol," in *MILCOM, Committee on Science, Space, and Technology Conference*. IEEE, 2010.

[21] G. Falco and N. Boschetti, "A Security Risk Taxonomy for Commercial Space Missions," in *ASCEND*, 2021, p. 4241.

[22] 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*, 2020.

[23] J. Pavur and I. Martinovic, "The Cyber-ASAT: On the Impact of Cyber Weapons in Outer Space," in *International Conference on Cyber Conflict*. NATO CCD COE, 2019.

[24] J. Drmola and T. Hubik, "Kessler Syndrome: System Dynamics Model," *Space Policy*, 2018.

[25] E. Jedermann, M. Strohmeier, M. Schäfer, J. Schmitt, and V. Lenders, "Orbit-based authentication using TDOA signatures in satellite networks," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2021.

[26] M. Qi and J. Chen, "An Enhanced Authentication with Key Agreement Scheme for Satellite Communication Systems," *International Journal of Satellite Communications and Networking*, 2018.

[27] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal War in Memory," in *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[28] G. Hunt, G. Letey, and E. Nightingale, "The Seven Properties of Highly Secure Devices," *Tech. Rreport MSR-TR-2017-16*, 2017.

[29] C. Li, Y. Zhang, R. Xie, X. Hao, and T. Huang, "Integrating Edge Computing into Low Earth Orbit Satellite Networks: Architecture and Prototype," *IEEE Access*, 2021.

[30] R. Morrison. (2022) Data Centres in Space Will Boost Satellite Computing Power and Storage. [Online]. Available: https://techmonitor.ai/technology/data-centre/data-centres-space-satellite-computing

[31] B. Nussbaum and G. Berg, "Cybersecurity Implications of Commercial Off The Shelf (COTS) Equipment in Space Infrastructure," *Space infrastructures: From risk to resilience governance*, 2020.

[32] ESA. (2019) Sandbox Satellite to Test Operations Innovations in Space. [Online]. Available: https://www.esa.int/Enabling_Support/Operations/Sandbox_satellite_to_test_operations_innovations_in_space?fbclid=IwAR14Asw229u5rsCNfxHKbLPbaZ0Gk9Ryq6QrEy_-4eAuRnVxnZCHRMhtflI

[33] R. Zheng. (2015) UFFS: Ultra-low-cost Flash File System. [Online]. Available: https://github.com/rickyzheng/uffs

[34] S.-. Team. (2017) SUCHAI Cubesat Flight Software. [Online]. Available: https://github.com/spel-uchile/SUCHAI

[35] UPSat Team. (2016) UPSat - The First Open Source Satellite. [Online]. Available: https://upsat.gr/

[36] Microsoft Azure. (2022) OreSat Firmware. [Online]. Available: https://github.com/oresat/oresat-firmware

[37] V. Singh, A. Prabhakara, D. Zhang, O. Yağan, and S. Kumar, "A Community-driven Approach to Democratize Access to Satellite Ground Stations," *GetMobile: Mobile Computing and Communications*, 2022.

[38] satnogs. (2022) SatNOGS Frontpage. [Online]. Available: https://satnogs.org/

[39] A. I. Support. (2022) Use Emergency SOS via satellite on your iPhone 14. [Online]. Available: https://support.apple.com/en-us/HT213426

[40] T. Scharnowski, N. Bars, M. Schloegel, E. Gustafson, M. Muench, G. Vigna, C. Kruegel, T. Holz, and A. Abbasi, "Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing," in *USENIX Security Symposium*, 2022.

[41] furryne and Ricky. (2018) Considering Using UFFS In Mission Critical Application. [Online]. Available: https://groups.google.com/g/uffs/c/6pBKhZq-FS0

[42] I. Sünter, "Design and Characterisation of Subsystems and Software for ESTCube-1 Nanosatellite," Ph.D. dissertation, Tartu University, 2019.

[43] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, "The Matter of Heartbleed," in *Internet Measurement Conference*, 2014.

[44] O. Koudelka, M. Wittig, and D. Evans, "ESA's OPS-SAT Nanosatellite Mission-A Laboratory in the Sky," in *IAA Symposium on Small Satellites and Earth Observation*, 2015.

[45] D. Evans and M. Merri, "OPS-SAT: A ESA Nanosatellite for Accelerating Innovation in Satellite Control," in *SpaceOps Conference*, 2014.

[46] W. Zhang, Y. Xue, J. Wu, and X. Xu, "Satellite as a Service: A Hybrid Resource Management Framework for Space-terrestrial Integrated Networks," in *IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2020.

[47] M. Quadrini, "Development of a Testing Emulation Platform for the Validation and Design of 5G Satellite Services," in *International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*. IEEE, 2021.

[48] CySec. (2022) Hack CySat. [Online]. Available: https://hack.cysat.eu/

[49] GomSpace. (2022) NanoMind A3200. [Online]. Available: https://gomspace.com/shop/subsystems/command-and-data-handling/nanomind-a3200.aspx

[50] M.-M. Didelot. (2021) How I Hacked an ESA's Experimental Satellite. [Online]. Available: https://www.deadf00d.com/post/how-to-hack-an-esa-experimental-satellite.html

[51] M. Pikelj and R. Heinrich. (2017) Successful Launch of German Technology Mini Satellite. [Online]. Available: https://www.airbus.com/en/newsroom/press-releases/2017-07-successful-launch-of-german-technology-mini-satellite

[52] J. Eickhoff, B. Chintalapati, P. Stoecker, W. von Kader, R. Traussnig, C. Sayer, R. Peel, and M. Keynes, *The Flexible LEO Platform for Small Satellite Missions*. Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2018.

[53] U. D. of Homeland Security. (2012) The Menlo Report. [Online]. Available: https://www.caida.org/publications/papers/2012/menlo_report_actual_formatted/

[54] S. S. Saha, S. Rahman, M. U. Ahmed, and S. K. Aditya, "Ensuring Cybersecure Telemetry and Telecommand in Small Satellites: Recent Trends and Empirical Propositions," *IEEE Aerospace and Electronic Systems Magazine*, 2019.

[55] S. Spinsante, F. Chiaraluce, and E. Gambi, "Evaluation of AES-based Authentication and Encryption Schemes for Telecommand and Telemetry in Satellite Applications," in *SpaceOps Conference*, 2006.

[56] GomSpace. (2022) Investor Presentation: Q1 Result and New Long-term Strategy. [Online]. Available: https://gomspace.com/UserFiles/Invester%20relations/investor_presentation_may2022_ABG_3-5-2022.pdf

[57] diamondo25. (2020) XTEA Encrypt Packet Nonce too Predictable. [Online]. Available: https://github.com/libcsp/libcsp/issues/162

[58] thusoy. (2020) MAC Comparison Leaks Timing Data. [Online]. Available: https://github.com/libcsp/libcsp/issues/44

[59] J. De Claville Christiansen, Y. SHOJI, and LibCSP Contributors. (2015) Cubesat Space Protocol - A Small Network-layer Delivery Protocol Designed for Cubesats. [Online]. Available: https://github.com/libcsp/libcsp

[60] D. Lucia, sv5d, and L. Bremond. (2022) Open Source Java Implementation of Publicly Available CCSDS Standards. [Online]. Available: https://github.com/dariol83/ccsds

[61] J. Pavur, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "A Tale of Sea and Sky on the Security of Maritime VSAT Communications," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2020.

[62] J. Pavur, D. Moser, V. Lenders, and I. Martinovic, "Secrets in the Sky: On Privacy and Infrastructure Security in DVB-S Satellite Broadband," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[63] G. Giuliari, T. Ciussani, A. Perrig, and A. Singla, "ICARUS: Attacking Low Earth Orbit Satellite Networks," in *USENIX Annual Technical Conference (ATC)*, 2021.

[64] J. Pavur, M. Strohmeier, V. Lenders, and I. Martinovic, "QPEP: An Actionable Approach to Secure and Performant Broadband from Geostationary Orbit," *Symposium on Network and Distributed System Security (NDSS)*, 2021.

[65] G. Oligeri, S. Sciancalepore, and R. Di Pietro, "GNSS spoofing detection via opportunistic IRIDIUM signals," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2020.

[66] D. Livingstone and P. Lewis, *Space, the Final Frontier for Cybersecurity?* Chatham House. The Royal Institute of International Affairs, 2016.

[67] M. Manulis, C. P. Bridges, R. Harrison, V. Sekar, and A. Davis, "Cyber Security in New Space," *International Journal of Information Security*, 2020.

[68] T. Harrison, K. Johnson, and T. G. Roberts, *Space Threat Assessment 2019*. Center for Strategic & International Studies., 2019.

[69] T. Harrison, K. Johnson, T. G. Roberts, T. Way, and M. Young, *Spacethreat Assessment 2020*. Center for Strategic and International Studies, 2020.

[70] D. P. Fidler, "Cybersecurity and the New Era of Space Activities," *Digital and Cyberspace Policy Program*, 2018.

[71] D. Barnard-Wills and D. Ashenden, "Securing Virtual Space: Cyber War, Cyber Terror, and Risk," *Space and Culture*, 2012.

[72] G. Falco, "Job One for Space Force: Space Asset Cybersecurity," *Belfer Center for Science and International Affairs, Harvard Kennedy School*, 2018.

[73] L. Yang, X. Cao, and J. Li, "A New Cyber Security Risk Eevaluation Method for Oil and Gas SCADA based on Factor State Space," *Chaos, Solitons & Fractals*, 2016.