# HoneySat: A Network-based Satellite Honeypot Framework

Efrén López-Morales[†,*] Ulysse Planta[◇,*] Gabriele Marra[◇], Carlos González[§‡], Jacob Hopkins[†], Majid Garoosi[◇], Elías Obreque[¶], Carlos Rubio-Medrano[†], Ali Abbasi[◇]

[†]*Texas A&M University–Corpus Christi* [◇]*CISPA Helmholtz Center for Information Security*
[§]*German Aerospace Center (DLR)*
[‡]*Departamento de Ingeniería Informática, Universidad de Santiago de Chile*
[¶]*Universidad de Chile*

## Abstract

Satellites are the backbone of several mission-critical services, such as GPS that enable our modern society to function. For many years, satellites were assumed to be secure because of their indecipherable architectures and the reliance on security by obscurity. However, technological advancements have made these assumptions obsolete, paving the way for potential attacks, and sparking a renewed interest in satellite security. Unfortunately, to this day, there is no efficient way to collect data on adversarial techniques for satellites, which severely hurts the generation of security intelligence.

In this paper, we present HoneySat, the first high-interaction satellite honeypot framework, which is fully capable of convincingly simulating a real-world CubeSat, a type of Small Satellite (SmallSat) widely used in practice. To provide evidence of the effectiveness of HoneySat, we surveyed experienced SmallSat operators currently in charge of active in-orbit satellite missions.

Results revealed that the majority of satellite operators (71.4%) agreed that HoneySat provides realistic and engaging simulations of CubeSat missions. Further experimental evaluations also showed that HoneySat provides adversaries with extensive interaction opportunities by supporting the majority of adversarial techniques (86.8%) and tactics (100%) that target satellites. Additionally, we also obtained a series of real interactions from actual adversaries by deploying HoneySat on the internet over several months, confirming that HoneySat can operate covertly and efficiently while collecting highly valuable interaction data.

## 1 Introduction

Artificial satellites are complex devices designed to withstand outer space conditions. They serve multiple purposes or types of *missions* that include position and navigation, e.g., the Global Positioning System (GPS) constellation [74], Earth observation, e.g., the Sentinel constellation [19], and broadband internet service, e.g., the Starlink constellation. In addition, Spacecraft can range from being as massive as thousands of kilograms, such as the International Space Station (ISS) to one kilogram CubeSats. As a result, the software and hardware components that make up a specific spacecraft vary greatly. Likewise, satellite missions' vary a lot depending on the owner's budget. For example, university missions are smaller as they have a limited budget. A cyberattack on a satellite or satellite constellation (group of satellites) could have disastrous consequences on a global scale, which is difficult to comprehend. Such an attack could lead to the cessation of air traffic and widespread communication blackouts. It could also cause food shortages and the freezing of financial transactions [11]. Furthermore, such an attack could exacerbate the Kessler Syndrome [18], a scenario in which collisions between satellites and debris in orbit create a cascade effect, generating even more debris, jeopardizing future satellite launches and operations.

In this increasingly vulnerable environment, the probability of a successful satellite cyberattack continues to rise. This is driven by three key trends [28]: first, satellite deployments have increased at an unprecedented pace. For instance, while an average of 82 launches took place between 2008 and 2017, as many as 197 launches occurred in 2023 alone, each typically carrying multiple satellites [61]. This surge is partly fueled by the rise of cheaper commercial off-the-shelf (COTS) components and the availability of more Space Launch Vehicles (SLVs), which makes access to orbit affordable for smaller institutions, such as universities [67]. Second, ground station technology has become significantly more affordable (and sometimes open source), greatly lowering the communication barrier with satellites [10]. Thus, a broader range of malicious actors can now communicate with satellites. Third, satellite engineers and operators continue to rely on *protocol obscurity* practices, such as hiding specialized knowledge about the transmission protocol implemented on satellites [81].

Although satellite security research has gained increased attention [53, 60, 79, 81], the relationship between outer space

[*]Both authors contributed equally to this work.

and cyberspace remains poorly understood [52]. At the same time, space-focused threat intelligence remains sparse, and our current methods for identifying tactics, techniques, and procedures (TTPs) used against these critical systems are limited. MITRE ATT&CK currently tracks 152 threat groups but shows only one that targets satellites explicitly [71], highlighting a significant data gap. Although the volume of reported cyber incidents in the space sector has grown, these reports rarely provide sufficient detail [11]. As a result, the security community has limited visibility into adversarial activity aimed at space infrastructure.

Honeypots' ability to collect real-world cyberattack data makes them an ideal solution to this problem [29]. A honeypot is a decoy computer system intended to lure and entice malicious actors to interact with it [12]; all the while, the honeypot logs all the interactions the attackers make. This log data can later be analyzed to discover new and existing TTPs.

Since the release of the first honeypot, the Deception Toolkit, in 1997 [13], a wide range of honeypots with ever-increasing capabilities have been introduced. These honeypots are used by universities, companies, and nation-states worldwide [8, 20, 27, 37, 62]. Honeypots are also used to deter malicious actors from attacking different types of systems, from industrial control systems [37] to social media platforms [3]. However, as of the time of writing, *there is no space-sector specific honeypot* in the literature.

In this paper, we design, implement, and test the first satellite honeypot, *HoneySat* to attract and analyze adversaries who attack space infrastructures over the Internet, a commonly observed threat vector [5, 7, 33, 80]. HoneySat is a modular, high-interaction honeypot framework that realistically simulates a complete satellite system (ground infrastructure and Satellite). Specifically, HoneySat simulates *Small Satellites* or *SmallSats* which are spacecraft with a mass of less than 180 kilograms [44]. CubeSats for example, are SmallSats. Additionally, as part of HoneySat, we developed the *Satellite Simulator*, a Python project to provide generic simulation functionality for users to populate satellite honeypots with believable data. This capability enables the creation of complete testing environments for satellite software that integrate a variety of simulated subsystems and sensors. For instance, Satellite Simulator can simulate orbital mechanics (e.g., position tracking, attitude adjustment) and electrical power systems (e.g., power generation, consumption, and distribution) alongside other subsystems.

We leveraged our framework to create honeypots of real-world CubeSat missions. Our results, backed by our survey of satellite operators, show that HoneySat's simulation is highly realistic. Our framework is able to simulate an entire real satellite mission, provides realistic telemetry, and supports real telecommands.

In summary, this paper makes the following contributions:

- A novel framework, HoneySat, a high-interaction, extensible honeypot for small satellites (Sec. 4).
- The Satellite Simulator, that simulates the physical processes, sensors, and subsystems necessary for a realistic satellite honeypot (Sec. 5).
- The results of a survey of experienced satellite operators that provide valuable insights into how realistically our honeypot performs, as well as experimental evidence demonstrating that the HoneySat framework can simulate small satellites, collect rich real-world interaction data, and be customized for multiple satellites (Sec. 6).

## 2 Background

This section lays out key background concepts that are relevant to satellite honeypots. For honeypots: a description of the different existing types (Sec. 2.1), as well as the current state-of-the-art (Sec. 2.2). For satellites: their operation (Sec. 2.3), their architecture (Sec. 2.4), existing Protocol Ecosystems (Sec. 2.5), and tactics, techniques, and procedures (TTPs, Sec. 2.6).

### 2.1 Types of Honeypots

Honeypots are categorized by interaction levels according to the interaction opportunities they provide. The two main types of honeypots are low-interaction and high-interaction.

**Low-Interaction Honeypots**. These honeypots offer minimal interaction, simulating real systems through scripts or finite-state machines. Their advantages are ease of setup and maintenance due to low resource consumption, and a reduced risk of adversarial takeover. However, they provide limited interaction opportunities to adversaries which limits the interaction data they provide. Low-interaction honeypot examples include Conpot [78] and Honeyd [54].

**High-Interaction Honeypots**. These honeypots offer extensive interaction opportunities via emulation or advanced simulations [54]. Their main advantage is providing adversaries with almost limitless interactions, enabling them to provide extensive interaction data. However, they pose a high risk of adversarial takeover as adversaries have more opportunities to hijack the honeypot [39]. High-interaction honeypot examples include Cowrie [50] and HoneyPLC [37].

### 2.2 Honeypot's State of the Art

The literature on honeypots includes hundreds of implementations that simulate a diverse set of computer systems [30, 48]. From classic implementations that simulate a host's TCP/IP stack, such as Honeyd [54], to modern approaches that integrate social media applications, such as HoneyTweet [3]. However, there is no satellite honeypot in the literature. In the absence of a satellite honeypot, we now examine the honeypot approaches most related to satellites: Industrial Control

Table 1: Comparison of Existing Honeypots and HoneySat.

Keys: ✓ = Supported; × = Not Supported.

| Honeypot/ Feature | Interaction Level | Included Protocols | Physics Sims | Extensibility |
|---|---|---|---|---|
| Conpot [78] | Low | 9 | 0 | ✓ |
| HoneyPLC [37] | High | 3 | 0 | ✓ |
| ICSPot [14] | High | 4 | 1 | × |
| HoneyICS [38] | High | 2 | 1 | ✓ |
| HoneyDrone [16] | Medium | 4 | 1 | × |
| **HoneySat** | High | 4 | 6 | ✓ |
| Addressed in Section | 4.4, 4.3 | 4.3, 5.2.1 | 4.4, 5.1 | 6.5 |

Systems (ICS) [35,36] and Unmanned Aerial Vehicles (UAV), a.k.a., *drones* summarized in Table 1.

Satellite systems like ICS must be aware of some physical process, e.g., its position, the sun's position, etc, via *sensors* to acquire data about the physical world. Several ICS honeypots have simulated these physical processes. For example, ICSnet [58] and HoneyICS [38] simulate several components such as programmable logic controllers (PLCs), and actuators such as water valves.

HoneyDrone [16] is a UAV honeypot that integrates different simulations, e.g., Ardupilot, to recreate multiple attack scenarios, including the UAV ground control station. Although UAV honeypots share some similarities with satellites [75], satellite honeypots require additional physical simulations and involve a more complex operation environment, which we discuss in Sec. 2.3.

## 2.3 Anatomy of a Satellite Mission

We now describe the components of a satellite mission. Due to satellite missions' complexity, we explain each component and match it with one of the numbers in Fig. 1. Every satellite mission includes the *ground segment* from which satellite operators control the satellite and the *space segment* which includes the satellite itself.

**Ground Segment** ①. The Ground Segment (GS) covers the terrestrial supporting infrastructure required for a successful satellite operation. It consists of a ground station, responsible for exchanging data with the spacecraft, the computational and network infrastructure required for communication, but also the systems to operate the satellites, e.g., servers, databases, and user interfaces [81]. Several ground stations can be connected in a network and coordinated as part of one GS. The GS includes the Ground Segment Software (GSS) that helps operators schedule and send commands and visualize data that is sent back in response.

**Space Segment** ②. The space segment comprises a satellite or a constellation of satellites. A satellite is launched into

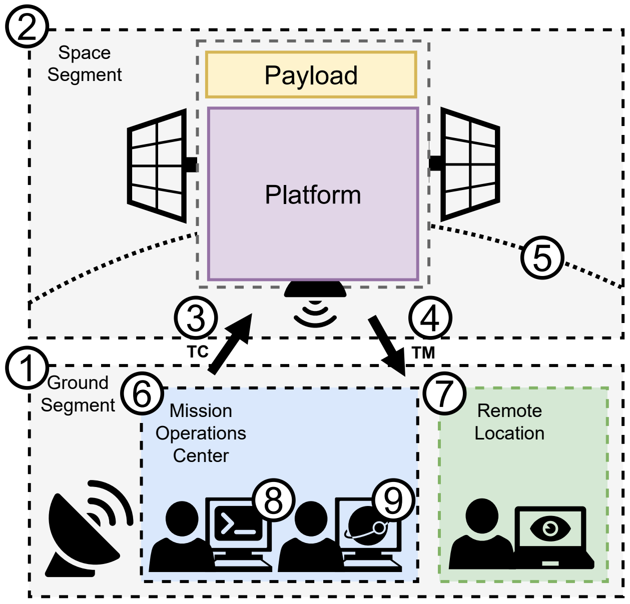


Figure 1: The components commonly found in a satellite mission in the context of the space and ground segments.

orbit and then establishes communications with the ground segment. During regular operations, satellites may communicate through one or multiple ground stations [81].

**Telecommands (TC)** ③ **and Telemetry (TM)** ④. The basic data flow between the space and ground segments are TC and TM [69]. TM is the data the satellite sends to the ground station which may contain the satellite's status, responses to previous commands, or payload data [69]. TCs are used to operate the satellite and are transmitted and encapsulated in a space protocol (see Sec. 2.5). The design and implementation of TCs varies depending on the satellite mission. From a security perspective, TCs are particularly important as an attacker that can send valid TCs to a satellite can fully take over the mission [81].

**Orbital Pass** ⑤. A satellite and its ground station can communicate *only* during an orbital pass. An orbital pass, or more commonly a *pass*, is when the satellite rises above a ground station's horizon and becomes available for communication. A pass's duration and timing depend on the satellite's orbit characteristics and any obstructing objects between the satellite and the ground station, e.g., mountains [82]. In some cases, such as geostationary orbit (GEO), the satellite can be above the horizon constantly. The occurrence of passes can be calculated using the spacecraft's orbital elements. A common representation of the orbital elements is the two-line element set (TLE) [77].

**Satellite Mission Operations** ⑥. Satellite mission operations vary widely depending on the owning organization, budget, and technology. Nevertheless, they share some commonalities, which we now describe.

A mission's operation involves a team of operators that use GSS to operate a satellite or a satellite constellation and ensure the mission's success [46]. Operations are carried out in a Mission Operations Center (MOC), where satellite operators sit at their workstations to manage TCs sent to the satellite(s).

Satellite operators may also remotely operate satellites ⑦ by connecting to the ground segment using tools such as desktop sharing like VNC (Virtual Network Computing) [10], or even rely on autonomous or semi-autonomous operations [26].

Satellite operations include two main activities: satellite tracking and TC generation and scheduling. Satellite tracking calculates the satellite's position in orbit and controls the ground station(s)' tracking antenna to establish communication between the ground and space segments. TC generation and scheduling crafts command to be programmed and sent to the satellite so it can perform different functions, e.g., read systems status data, schedule orbital maneuvers, schedule payload operations, and download payload data.

**Ground (Segment) Software (GSS)**. GSS allows satellite operators to carry out the satellite mission's routine operations as described in the previous paragraph. GSS are very diverse. Some satellite missions develop their own GSS while others use open-source [65] or buy proprietary GSS like GSWeb [23]. There are two main types of GSS: Mission Control Software (MCS) and Ground Station Control Software (GSCS).

Mission control software ⑧ manages TCs and scripts to be sent to the satellite and can display TM. For example, ESA's SCOS-2000 is an MCS that provides generic functionality that can be customized for a specific mission to cover the functions required for TM reception and processing and TC verification [42]. Some satellite missions develop their own MCS. For example, the SUCHAI mission team developed their MCS application to send and receive TC/TM [65].

Ground station control software (GSCS) ⑨ helps satellite operators track and visualize the satellite's orbit and provide detailed information about each satellite's pass. For example, Gpredict is a popular open-source GSCS that performs real-time satellite tracking and orbit prediction [15]. Gpredict can also interface with and control the radio system and antenna rotor during the satellite pass.

## 2.4 Satellite Architecture

Satellite architectures are varied and complex, however, here we describe the most common terminology depicted in Fig. 1's space segment ②.

When referring to satellite architecture there is a distinction between the *platform* that facilitates the successful operation of the satellite's day to day-to-day activities and the *payload*. The platform provides the possibility to run a payload that fulfills the purpose of these missions. This payload differs based on the goal of the mission and can range from instruments to measure physical properties to communications systems providing wireless connectivity.

**Platform**: The platform, or satellite bus, is a system composed of custom-designed or off-the-shelf subsystems necessary for critical satellite operations. These include, among others, the Attitude Determination and Control System (ADCS) to maintain the satellite's orientation (i.e. attitude) and position to keep the satellite pointed towards antennas or solar panels illuminated; the Electrical Power Subsystem (EPS) for managing power generation and distribution; the Communication Subsystem (COMM) and the Command and Data Handling (C&DH) subsystems to facilitate communications for receiving TC and sending TM and controlling the satellite operations.

All of these subsystems are then controlled via TCs sent to the satellite. TCs may be, depending on the protocol ecosystem and complexity of the subsystem, interpreted by a central C&DH System or merely forwarded to the recipient subsystem for further processing [84]. This managerial duty is left to Flight Software (FS) running on an embedded system of differing complexity. Some examples include NASA's Core Flight System (cFS) [41] and F' (F Prime) [6], KubOS [83], the German Aerospace Center (DLR)'s OUTPOST [17], and the University of Chile's SUCHAI FS [24]. Attackers aim to gain the ability to send Telecommands to the Flight Software, as this typically grants control over all spacecraft subsystems.

**Payload**. The payload is the equipment that the satellite employs to fulfill its mission. Due to satellites' varied missions, payloads are heavily customized [70]. For example, if a satellite's mission is remote sensing, its payload may include an infrared camera [45]. craft's essential functions, acts on TCs, and handles internal data transmitted to and from other systems [84]. The CDHS is comprised of Flight Software (FS) running on an OBC [81].

## 2.5 Small Satellite Protocol Ecosystems

SmallSat missions can often be categorized into the following categories by the adoption of protocols and their corresponding philosophies.

**Cubesat Space Protocol.** The Cubesat Space Protocol (CSP) family of protocols is a one-stop solution for Small missions [63]. There are not a lot of choices left to the operator/developer and there are two vendors mainly offering components using this protocol and some that offer compatibility solutions [23]. If a mission built on CSP is expanded by a commercial subsystem it will plug in without issue after configuration.

CSP is implemented as an open-source C library called *libCSP* [63]. CSP follows the TCP/IP model, including transport and routing protocols and multiple layer 2 interfaces such as I2C (Inter-Integrated Circuit), CAN (Controller Area Network), and ZeroMQ (ZMQ) for transmission on TCP/IP networks [72].

In CSP the ground segment interfaces and satellite subsystems are part of a CSP network. Sending TC is as simple as sending a CSP packet with an address corresponding to a node that is part of the satellite. The configured static routing tables of the nodes on the network will then cause the packet to be passed to the ground station and transmitted to the satellite over RF where a system on board will route the packet to its destination. In case the satellite is not currently doing a pass, the packet is dropped.

The straightforwardness of CSP makes it particularly interesting for the honeypot use case as the internal structures of any CSP-based mission will look very similar, so distinguishing two missions is difficult as they will use the protocols in a very similar way.

There are also default services running on standardized ports that are enabled by default when building libCSP [63], for example for network diagnostic (ping) or basic operations (rebooting).

In CSP missions, nodes on the network typically feature a command-line-based interface for configuration and debugging purposes [23]. This may allow someone with access to it to interact with the current node and other nodes on the network.

**CCSDS Space Communication Protocols.** The CCSDS Space Communication protocols are a vast set of standardized protocols used for different purposes in space communications. A major CCSDS protocol relevant for SmallSat TM and TC is called spacepacket. This protocol is used in combination with the ECSS Packet Utilization Standard (PUS) to define how TCs and TM are encoded and transported. The PUS defines services (and thus sets of TC/TM) for functionality that satellite missions likely require, including large data transfer or event reporting [59]. Moreover, mission designers can tailor the PUS standard by selecting a subset of services and sub-services relevant to their mission needs. Furthermore, it is possible to define custom Services or implement numerous custom functionalities.

**Proprietary Protocols.** In addition to these widespread ecosystems, vendor-specific sets of protocols may be usable on top of more standard protocols or transport higher-layer packets of known protocols. Some basic small satellites may also have entirely ad hoc protocols that are limited to use with a single mission or a set of satellites by a specific institution. These proprietary protocols are not ideal candidates for constructing honeypots as an ecosystem of one mission that is unique to a small set of missions will look out of place for other missions, preventing generalization.

The choice of ecosystem also influences the choice of GSS as an MCS usually focuses on a single protocol stack. In addition to the space protocol ecosystems above, satellite operators use well-known network protocols to connect with the mission's ground segment services. These protocols include Telnet, SSH, VNC, FTP, and HTTP. Telnet, SSH, and VNC are used to remotely connect to the MOC workstations. HTTP is used for web interfaces that operators use to visualize mission data, and FTP is used to download TM data from the mission data repository. If an adversary were to compromise any of these services, they could use it as a stepping stone to target the satellite itself.

## 2.6 Space Systems' Tactics, Techniques and Procedures (TTPs)

Tactics, Techniques, and Procedures (TTPs) describe the behavior of a malicious actor in a structured scheme to understand how they might execute an attack [47, 56]. Several frameworks have been introduced to standardize space systems' TTPs. There are two MITRE-style frameworks, the SPARTA matrix [68] and the SPACE-SHIELD matrix [4]. These matrices list and describe space security-specific tactics and techniques such as *initial access* and *ground segment compromise*.

## 3 Threat Model

Following Fig. 1, we assume an adversary willing to compromise a satellite can only interact with the space segment simulation by sending TCs from the ground segment first. To gain initial access to the ground segment, an adversary needs to connect via one of exposed network protocols depicted in Sec. 2.5, namely, VNC, Telnet, TCP/IP, etc., which correspond with the operational protocols used in real satellite missions.

From there, an adversary may try to launch different commands to take full control and/or compromise the services offered by the satellite's mission. Finally,

in this paper, the modeling of physical radio communication between the space and ground segments, which is commonly used in practice, is considered out of scope and left for future work.

## 4 HoneySat Framework Design

In this section, we explain the objectives we aim to achieve (Sec. 4.1), the design principles we follow to meet such objectives (Sec. 4.2), and the overall design of our Space (Sec. 4.4), and Ground (Sec. 4.3) segments.

## 4.1 HoneySat's Design Objectives

Our design aims to achieve the following objectives:

DO-1 **Capability to Capture Rich Interaction Data.** As explained in Sec. 1, the number one objective of any honeypot is to capture interaction data from which we can derive knowledge on adversaries' TTPs. As such, our first objective for HoneySat is to have capability to capture rich interaction data.

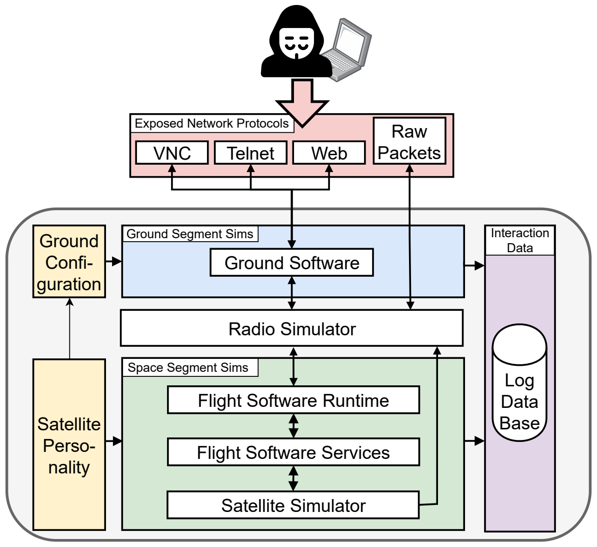


Figure 2: HoneySat framework high-level architecture design.

DO-2 **Provide Deception.** As we discussed in Sec. 1, honeypots' nature must remain *covert* to entice adversaries into interacting with it. As such, our second objective is for HoneySat's nature to remain hidden from adversaries.

DO-3 **Provide Extensibility and Customizability.** A framework's main purpose is to provide generic functionality that can be customized to meet the user's needs, in HoneySat's case, we must be able to support multiple SmallSats. For example, a particular SmallSat may use CSP or CCSDS ecosystems. Additionally, each SmallSat mission has a particular orbit that must be customizable. As such, our third objective is for HoneySat to provide extensibility and customizability.

## 4.2 HoneySat's Design Principles

To meet our objectives, we selected the following principles:

DP-1 **High-Interaction Simulation.** As we described in Sec. 2.1, high-interaction honeypots give adversaries the same or almost identical interaction opportunities as a real satellite. Thus, they can log many of the TTPs discussed in 2.6. For these reasons, we selected high-interaction simulation as our first design principle. This design principle is based on design objective DO-1.

DP-2 **Realistic Simulation.** As we discussed in Sec. 2.2, satellites keep track of their orbit location, the position of the sun, among others. These details must be simulated by our honeypot; otherwise, they may alert adversaries that they are interacting with a fake system. This design principle is based on design objective DO-2.

DP-3 **Modularity.** As we explained in Sec. 2, satellites are complex and diverse systems. There is no one-size-fits-all solution. To tackle this problem, we designed HoneySat to be modular. Modularity allows us to insert, remove, or change any part of our honeypot framework without implementing a new honeypot from scratch. This design principle is based on our objective DO-3.

| Segment | Real Component | HoneySat Component |
|---|---|---|
| Ground | Remote Operation Protocols | VNC, Telnet |
| Ground | Mission Control Software | Modified MCS |
| Ground | Ground Station Control Software | Gpredict |
| Ground | Web Interface | Web Interface |
| Ground | Ground Station | Radio Simulator |
| Space | Flight Software | F.S. Runtime + Services |
| Space | On Board Computer | Docker Container |
| Space | Platform | Satellite Simulator |
| Space | Payload | Satellite Simulator |

Table 2: Correspondence between real satellite mission components and HoneySat's design components.

We now describe HoneySat's architecture design and how this architecture integrates the *design principles* described above. At the highest level, our framework's architecture, depicted in Fig. 2, consists of two sets of simulations, the *space segment simulations* and the *ground segment simulations*. Table 2 illustrates how HoneySat's simulation components match the components of a real satellite mission.

## 4.3 Ground Segment Design

Following Sec. 2.3, the purpose of the HoneySat's ground segment is to simulate the ground segment assets, e.g., the ground segment software. To accomplish this, our design includes the following components: 1) the *Exposed Network Protocols*, 2) the *Ground Segment Software*, 3) the *Radio Simulator*, 4) the *Ground Personality*, and 5) the *Logging Repository*.

**1) Exposed Network Protocols.** To provide adversaries with feasible access to our honeypot, HoneySat exposes multiple means of interaction over a network such as the Internet. We designed HoneySat to support four different interaction methods using different protocols, namely VNC, Telnet, Web, and Access to the ground station via raw packet transmitting capability (following design principle DP-3).

**2) Ground Software.** As discussed in Sec. 2.3, the ground software includes mission control software (MCS) and ground station control software (GSCS). We leverage existing ground software, allowing HoneySat to provide a high-interaction simulation (DP-1).

**3) Radio Simulator.** As discussed in Sec. 2.3, communication between the satellite and the ground station is possible only during a pass. The purpose of the Radio Simulator is to control communication between the ground and space segment simulations to simulate real orbital passes. The Radio Simulator design follows design principles DP-2 and DP-3.

**4) Ground Configuration.** The ground configuration is a series of settings for the various ground segment-specific configurations, e.g., the satellite mission logo in the Web Interface.

**5) Logging Repository.** This repository logs data such as the TM/TC traffic to and from the ground station software and the logging attempts received in the web interface. We designed the ground segment simulation logs to be categorized and timestamped. The logging design follows design principle DP-3.

## 4.4 Space Segment Design

The purpose of the HoneySat's space segment is to mimic the spacecraft, as discussed in Sec. 2.3. To accomplish this, our design includes the following components: 1) the satellite's *Flight Software Runtime*, 2) the *Satellite Simulator*, 3) the *Flight Software Services*, 4) the *Satellite Personality*, and 5) the *Logging Repository*.

**1) Flight Software Runtime.** As discussed in Sec. 2.4, the satellite flight software manages all critical functions required for the mission operation, such as interacting with hardware peripherals, processing TCs, and sending TM. For this to work we need an environment where services that handle TC intended to run on flight software can be run. We reuse existing compatibility or testing wrappers to run the relevant parts of flight software for HoneySat to provide nearly identical interactions to an adversary. This produces a high-interaction honeypot simulation that follows design principle DP-1.

**2) Satellite Simulator.** One of the biggest challenges when designing HoneySat was simulating all the physical processes, e.g., attitude, that satellites need to know. The Satellite Simulator is designed to solve this problem. It simulates all the necessary satellite subsystems, e.g., the Electrical Power System, and the sensors, e.g., a GPS receiver, that a satellite requires to function. The purpose of the Satellite Simulator is to abstract away the various hardware peripherals that communicate with the flight software by simulating these peripheral's data. For example, suppose a particular flight software was implemented to utilize a specific lithium-ion battery pack. In that case, the Satellite Simulator's power system simulation simulation can simulate the basic behavior of the battery. We designed the Satellite Simulator to be modular so that new subsystems and sensors can be easily added, modified, or disabled, following our DP-3.

**3) Flight Software Services.** The flight software services that process TC and provide TM act as the bridge between the Satellite Simulator and the rest of the Honeypot. In case a service requires some data from a subsystem of the satel-

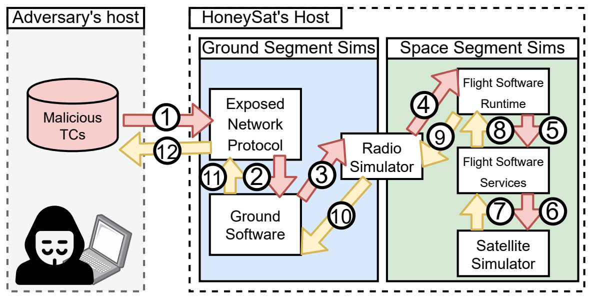


Figure 3: HoneySat's Theory of Operation.

lite or a command is sent to a subsystem, it is passed to the satellite simulator instead. The *satellite state module* routes the service's messages to and from the Satellite Simulator. It uses a list of message IDs that can be customized based on the protocol ecosystem and software architecture.

**4) Satellite Personality.** The satellite personality module is designed to provide a central configuration location for the space segment. It includes FS and Satellite Simulator configuration values such as the satellite's battery capacity. The satellite personality allows our framework to be easily customized.

**5) Logging Repository.** We designed each space segment component to provide detailed logs. As shown in Fig. 5, all Satellite Simulator modules can send their own logs.

## 4.5 Theory of Operation

In this section, we provide a brief overview of how the designed framework components shown in Figure 2 interact by following the numbers in Fig. 3.

An adversary gets initial access via one of the *Exposed Network Protocols* ①. On interaction with the *Simulated Ground Segment*, an attacker is presented with access to the *Ground Software* ②. The configuration of this ground software is defined by the *Ground Configuration*, which allows the ground software to look like one of many different missions from its protocol ecosystem to the attacker. The attacker can then interact with the *Ground Software*, while their actions are reported to the *Log Database*. They might want to try to gain more privileges on the ground segment or try to send TC to the space segment. Instead of deploying a ground station with an RF transmitter and associated hardware, we employ the *Radio Simulator* ③ to handle all simulated radio frequency communications.

When an attacker uses the ground segment to send a valid TC or raw packets, the *Radio Simulator* checks whether the satellite configured in the *Satellite Personality* is currently passing over the designated ground station location. If so, the *Radio Simulator* forwards the TC to the *Flight Software Runtime* ④. When the *Flight Software Runtime* receives a command, it will run the corresponding *Flight Software Service* to compute a response ⑤. In case it requires either changes to an on-board system's state or a sensor value to execute the TC, it will invoke the *Satellite Simulator* for this ⑥. The *Satellite Personality* contains the configuration used by the *Satellite Simulator*. Once a TC is processed, the resulting TM is routed back through the *Radio Simulator* to either the attacker or the ground software used by the attacker ⑦-⑫, and is also recorded in our *Log Database*.

## 5 HoneySat's Implementation

Having laid out HoneySat's design we now explain how we implemented the Satellite Simulator (Sec. 5.1), the generic CSP mission honeypot (Sec. 5.2), the ground segment (Sec. 5.2.1), and the space segment (Sec. 5.2.2).

### 5.1 Satellite Simulator Implementation

We implemented the Satellite Simulator as a Python object-oriented programming application.

We implemented 11 sensors, 4 subsystems, 1 satellite state, and 1 interface, based on our 6 simulations. These modules are generic enough so that we do not have to heavily modified them to support other satellites. Overall, satellite architectures, subsystems, and sensors are varied, so modularity is necessary when constructing a simulation framework. Many of the basic simulations and corresponding sensors can be shared, especially in a honeypot where attackers lack detailed information that would enable them to distinguish between small differences in sensor values. Further details on the simulations can be found in A.2

### 5.2 Generic CSP Mission Honeypot

Following our design we implemented HoneySat to support the CSP ecosystem. This implementation provides a generic CSP-based mission honeypot that can be customized to simulate a specific CPS-based satellite missions.

#### 5.2.1 Ground Segment Implementation

We now describe how we implemented the five ground segment simulations designed in Sec. 4.3. 1) the *Exposed Entry Points*, 2) the *Ground Software*, 3) the *Radio Proxy*, 4) the *Ground Configuration*, and 5) *Logging Repository*.

**1) Exposed Network Protocols.** We implemented four exposed entry points. A VNC server, a Telnet server, a Web server, and ZeroMQ-based access to the CSP Network. We implemented the VNC server using TigerVNC [51] and used PyZMQ [25] for the Raw Packet Access. The Telnet server was implemented using Python to expose the command-line interface of GSS on the network. The web interface allows for the customization of text presented to the attacker through

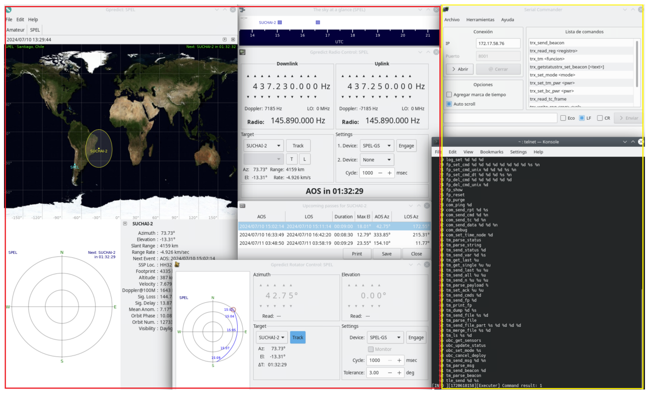


Figure 4: The view an attacker would see upon connection to the VNC server. Ground Station Control Software (Red) Ecosystem/Mission Specific MCS (Yellow).

configuration. Using the above entry points, adversaries can interact with the GSS and the space segment simulations. Among these, the web interface serves as a measure to attract attackers and serves as an initial entry point for attackers. We developed a simple yet convincing web application, presented as an early version of a web interface currently under development. After the login, the attacker would be welcomed by a dashboard that displays information about the satellite and the latest output of the Ground software command line interface. Additionally, the web application provides access to documentation and the VNC server (Fig. 4), facilitating the attacker in finding the other entry points.

**2) Ground Software.** To implement the MCS software for our CSP-based honeypot, we require an open-source MCS for the CSP ecosystem. We leveraged the SUCHAI mission control software (SUCHAI MCS). This software is a node on the CSP network that has a command line-based interface that allows basic MCS functionality like scheduling telecommands or saving downlinked telemetry. This is a good choice for the generic CSP MCS as it can be passed off as the CLI of commercial ground segment appliances like the GomSpace NanoCom MS100 [23]. In this generic honeypot, we can set mission-specific parameters through the ground configuration. Additionally, we provide a desktop environment accessible via VNC, complete with supporting applications such as configured Gpredict for the satellite mission. For certain deployments, this environment also includes a mission-specific GUI tool, both of which appear as open windows on the desktop.

**3) Radio Simulator.** The Radio Simulator was implemented as a *ZMQ Hub*. The ZMQ Hub functions as a router for the CSP network. Simulating both the ground part of the CSP network including the ground station and space segment. Depending on whether the satellite is currently passing, packets to the space segment are forwarded to the Flight Software Runtime. We decided to use ZeroMQ as it is the protocol to use for transmitting CSP over TCP/IP [64].

**4) Ground Configuration.** We implemented the ground configuration on each component, and the settings can be modified through a Docker compose file explained in Appendix A.3.

**5) Logging Repository.** This implementation uses the MongoDB database [43] to aggregate and store logs. The web interface records login attempts, while the Telnet server logs inputs on a per-connection basis. Additionally, network traffic to and from the host running HoneySat is captured.

#### 5.2.2 Space Segment Implementation

**1) Flight Software Runtime** Although most of the flight software frameworks presented in Section 2.4 are open-source projects, it is difficult to find a real satellite flight software implementation that is available for study and modification [81]. For this reason, we use the *University of Chile's SUCHAI Flight Software* (SUCHAI FS) [24], which has been deployed in four satellite missions [21]. The SUCHAI FS provides Flight software based on the CSP ecosystem that already features a mode where it can be run on a POSIX OS.

**2) Flight Software Services.** The SUCHAI FS uses libCSP [63] and thus features the libCSP default services that present a generic attack surface. Furthermore, there is a set of commands commonly used in satellites as a basic set of tasks to control common satellite operations such as communications. There are 96 of these commands and parameters that are parsed from CSP packets using a text-based protocol.

**3) Satellite Personality.** The satellite personality was implemented as a Python class that holds multiple variables related to the space segment. The current implementation includes almost 50 configurable variables, such as the satellite's name, the battery's nominal cell capacity, etc.

**4) Logging Repository.** Like the ground segment's logging repository (Sec 5.2.1), The Satellite Simulator and the FS are connected to the MongoDB instance using a MongoDB client.

Each logged data entry is created with customized tagging and a timestamp. The collected data is structured JSON [32] format and stored in the MongoDB database.

Finally, we containerized and hardened this version of the honeypot implementation for secure and easy deployment, which we describe in Appendix A.3.

## 6 Evaluation

Having described HoneySat's implementation in Sec. 5, we now evaluate it. This section lists three experimental questions designed to test HoneySat's alignment with our design objectives. Next, we present four sets of experiments that provide empirical evidence confirming that our design objectives have been achieved within HoneySat. We describe each experiment's environment, methodologies, and results.

### 6.1 Experimental Questions

These questions aim to determine whether or not HoneySat's meets the design objectives outlined in Sec. 4.1.

Q-1 **Can HoneySat offer extensive interaction opportunities to adversaries?**
Since capturing data on varied techniques is the purpose of any honeypot, we explore the capabilities of HoneySat, as described in Sec. 4. This question is related to design objective DO-1 and is addressed in Sec. 6.2 and 6.4.

Q-2 **Can HoneySat simulate a SmallSat mission well enough to deceive adversaries?**
After enticing an adversary HoneySat must keep its true nature hidden, we need to simulate the satellite's communication and physics characteristics described in Sec. 2.4. This question relates to design objective DO-2 and is answered in Sec. 6.3.

Q-3 **Can HoneySat simulate different SmallSat missions?**
HoneySat's customization is important because it would allow users of our framework to implement their own honeypots. This question relates to design objective DO-3, and we answer it in Sec. 6.5.

To answer the above research questions we conducted four experiments. First, in Sec. 6.2 we craft multiple attacks in a controlled environment to quantify the level of interaction that HoneySat provides. Second, in Sec. 6.3 we conduct a survey with experienced satellite operators to evaluate HoneySat's realism. Third, in Sec. 6.4 we deploy HoneySat and expose it to the Internet to test its deception capabilities. Fourth and final, in Sec. 6.5 we add an additional protocol ecosystem to HoneySat to test its extensibility potential.

### 6.2 TTP Interaction Experiment

This experiment seeks to answer Q-1 by quantifying the different interactions that HoneySat provides. To achieve this, we leveraged the SPACE-SHIELD matrix (Version 2.0) [4] discussed in Sec. 2.6. SPACE-SHIELD provides a collection of adversary tactics and techniques for space systems. In this experiment, we determined the number of tactics and techniques that HoneySat supports. SPACE-SHIELD consists of 14 tactics and 62 techniques. However, not all of them are applicable to a virtual, network-based honeypot such as HoneySat. For example, the technique *Compromise Hardware Supply Chain* involves "replacing a hardware component in the supply chain with a custom or counterfeit part" which is out of the scope of a virtual honeypot. Taking this into consideration, 14 tactics, and 33 techniques are applicable to HoneySat.

**Experiment Description.** Our experimental environment included two hosts. One host running HoneySat and the adversary host. The HoneySat host ran Ubuntu 23.10 and was configured with the SUCHAI-2 satellite and ground personalities. The adversary host ran a Telnet client. Both hosts were connected to the same network.

**Experiment Methodology.** We designed one interaction or exploit for each of the applicable 33 techniques for our honeypot. The interactions involved ground segment simulations such as the web interface. The exploits were simple, using one TC, or complex with multiple TCs involved.

**Experiment Results.** We crafted 16 exploits and 17 interactions following the above methodology. For example, the technique *T1007 - System Service Discovery*'s description reads "Adversaries may try to gather information about registered local system services. Adversaries may obtain information about services using tools and OS utility commands. Adversaries may use the information from System Service Discovery during automated discovery to shape follow-on behaviors, including whether or not the adversary fully infects the target and/or attempts specific actions."

Based on this description, we designed the following exploit to capture information about the satellite's running processes:

```
TC-1: 1: obc_system ps -aux > ps.log
TC-2: 1: tm_send_file 10 ps.log
```

Similarly, technique *T2014 - Backdoor Installation* reads "An attacker can interfere with the hardware or the software, integrating or modifying the existing software, hardware configuration, or the transponder configuration to permit himself future access to the resource". Therefore, we designed the following exploit to upload a malicious script to the satellite software:

```
TC-1: tm_send_file backdoor.sh 1
TC-2: 1: obc_system ./recv_files/backdoor.
         sh
```

Due to space limitations, we do not describe all the exploits and interactions here. However, the complete exploit and interaction list is available in Table 6 in Appendix A. The overall results for this experiment are depicted in Table 3. The key findings of our experiment are shown below, providing strong evidence for answering question Q-1 in the affirmative and design objective DO-1 as achieved.

**Key findings Q-1**

- HoneySat supports 86.8% of the SPACE-SHIELD matrix techniques possible in a virtual satellite honeypot.
- HoneySat supports 100% of the SPACE-SHIELD matrix tactics.

| Tactics | SPACE-SHIELD Techniques (Applicable to Virtual Honeypots) | HoneySat Supported Techniques |
|---|---|---|
| Reconnaissance | 2 | 2 |
| Resource Development | 2 | 2 |
| Initial Access | 2 | 2 |
| Execution | 2 | 2 |
| Persistence | 2 | 2 |
| Privilege Escalation | 2 | 1 |
| Defense Evasion | 4 | 4 |
| Credential Access | 3 | 3 |
| Discovery | 2 | 2 |
| Lateral Movement | 4 | 1 |
| Collection | 2 | 2 |
| Command & Control | 2 | 2 |
| Exfiltration | 2 | 1 |
| Impact | 7 | 7 |
| Total | 38 | 33 |

Table 3: Tactics and techniques supported by HoneySat compared to the ones included in the SPACE-SHIELD matrix that are applicable to virtual honeypots.

## 6.3 SmallSat Operators Survey

Evaluating the realism of a satellite honeypot is challenging for two main reasons. First, as we discussed in Sec. 1, satellites, including SmallSats are very diverse. Second, there is no established metric or tool, such as Nmap's OS detection [49], to quantify the level of realism of our honeypot.

To evaluate the realism and deception capabilities of HoneySat, we surveyed *experienced, real-world SmallSat operators*. Because operators interact with real-world SmallSat missions on a daily basis they are *experts* and thus are the best population to rigorously evaluate HoneySat.

### 6.3.1 Survey Structure

We divided our survey into five sections. The first section collected essential background information about the participants, while the second part focused on their professional experience. In this second section, participants were asked whether they worked in industry, government, or academia and to self-report their cybersecurity skills as well as satellite-related skills (e.g., satellite hardware engineering). The third section of the survey probed participants' satellite operation

| **Satellite Operation** | **Evaluated Component** | **No. Qts.** |
|---|---|---|
| Telecommand Scheduling | Mission Control SW | 6 |
| Pass Prediction | Ground Station Control SW | 6 |
| Antenna Positioning | Ground Station Control SW | 6 |
| EPS Telemetry | Satellite Simulator | 6 |
| Temperature Telemetry | Satellite Simulator | 6 |
| ADCS Telemetry | Satellite Simulator | 6 |
| RGB Camera Telecommand | Satellite Simulator | 6 |
| Ping Test | Radio Simulator | 6 |

Table 4: Summary of questions in Section 6.3, *satellite honeypot operation tasks* of the survey (48 total).

experience, including how many missions they had operated and which tools they used.

The largest section of the survey included 48 questions about *satellite honeypot operation tasks*. Participants watched eight 1–2 minute recordings of HoneySat 's VNC view (Fig. 4), each showcasing the ground and satellite personality of a real CSP-based CubeSat mission (based on the SUCHAI-2 CubeSat [34]). Every recording highlighted a different feature of HoneySat by replicating a real-world satellite mission operation, as discussed in Section 2. After each recording, participants answered questions designed to assess how realistic they found that particular aspect of the honeypot. Table 4 lists all of the satellite operations demonstrated in these recordings. Finally, once participants had seen multiple elements of our honeypot, the survey concluded with an overall evaluation section. Both the honeypot operation tasks and the overall evaluation used 5-point Likert scale [55] questions to measure participants' positive or negative reactions.

### 6.3.2 Participants

We conducted the survey via Qualtrics and distributed the survey directly to *active SmallSat mission operators* from previously identified missions. In total, we received responses from 14 satellite operators who have between 1 and 5 years of experience operating satellites and have operated between 2 and 7 unique missions. In terms of demographics, 21.4% (3/14) of participants were female, 71.4% (10/14) male and 7.1% (1/14) preferred not to answer. 57.1% (8/14) belonged to the 18-24 age group, 35.7% (5/14) to 25-34 and 7.1% (1/14) to 35-44. 28.5% (4/14) of participants' highest level of education was high school, 35.7% (5/14) bachelor's degree, 14.2% (2/14) master's degree and 21.4% (3/14) Ph.D. In regards to geographic location, 78.5% (11/14) of participants were located in Europe, 14.2% (2/14) in North America and 7.1% (1/14) in South America.

Recruiting participants was challenging due to the rarity and specialized nature of the required expertise. Over a span of four months, we reached out to national and international institutions, as well as private corporations involved in operational satellite missions to identify suitable participants.

### 6.3.3 Methodology and Key Results

In the survey, we aimed to evaluate three key aspects of our honeypot. First, whether the ground segment simulations are realistic; second, whether the space segment simulations are realistic; and third, whether HoneySat, as a whole, provides a convincing and realistic system.

Before describing the results it is important to emphasize that the participants were *informed* that the recordings showed a simulation of a satellite mission and not of a real mission.

**Ground Segment Realism.** To understand if HoneySat's ground segment is realistic we showed participants recordings of different satellite operations (discussed in Sec. 2.3) performed with HoneySat, to showcase the ground segment components listed in Table 4.

For example, one of the recordings shows the *pass prediction* operation which involves calculating when and where a satellite will be visible or within range of a specific ground station. This operation is performed using the ground station control software. After the participants watched a recording of this operation, we asked them to rate their perceived level of realism. 85.7% of participants agreed or strongly agreed that the *pass prediction* operation done in HoneySat resembles that of a real mission.

Another relevant operation is the *telecommand scheduling* operation which involves the planning and organization of the commands to be sent to the satellite during a pass. 42.8% of participants strongly agreed that the *telecommand scheduling* operation performed in HoneySat resembles that of a real mission mentioning the use of a terminal-based mission control software. Interestingly, 35.7% of participants neither agreed nor disagreed, citing that they do not use a command line tool but instead a GUI. These results indicate that some of the surveyed operators use a command line-based mission control software while others use a GUI-based confirming the diversity of means of operation in Sec. 2.3.

In summary, according to our survey, most participants perceive the ground segment simulated by HoneySat as comparable to that of a real satellite mission. While a few participants mentioned that they expected to see a GUI-based mission control software instead of a command line-based one, this is something that we expected, as the means of satellite operations vary greatly among missions. Nevertheless, thanks to HoneySat's modularity HoneySat can be extended to use a GUI-based mission control software such as YAMCS [1].

**Space Segment Realism.** In a similar manner, to understand if HoneySat's space segment is realistic, we showed participants recordings of different satellite operations that make use of HoneySat's space segment simulations (discussed in Sec. 4.4) to showcase multiple components of the space segment listed in Table 4. For example, one of the recordings shows the *EPS Telemetry* operation which involves the collection of data from the Electrical Power Subsystem discussed in Sec. 2.4. This operation is performed using the mission control software to send telecommands and download the latest EPS telemetry. After the participants watched a recording of this operation performed by HoneySat, we asked them to rate their perceived level of realism of the telemetry output shown during the operation. 57.1% of participants agreed or strongly agreed that the telemetry shown in the EPS Telemetry operation resembles that of a real mission mentioning realistic EPS values such as voltage, and temperature. 14.2% neither agreed nor disagreed mentioning that the EPS temeletry was presented in a different format. Finally, 28.5% disagreed or strongly disagreed mentioning that the telemetry included less values than the mission that they had operated. These results indicate that the EPS telemetry generate by the Satellite Simulator is considered realistic by the majority of participants.

We also asked participants to asses the *ping test* operation. This operation is a diagnostic procedure used to verify the communication link between a ground station and the satellite. It involves sending CSP packets between two CSP nodes, namely the ground segment and the spacecraft. After the participants watched a recording of this operation performed on HoneySat, we asked them to rate their perceived level of realism. 64.2% of participants agreed or strongly agreed that the ping test operation resembles that of a real mission citing the realistic response times. 21.4% of participants neither agreed nor disagreed stating that in their experience the ping test also downloads additional telemetry. Finally, 14.2% disagreed or strongly disagreed noting that in their mission the ping command has a different name. The results of the *ping test* operation indicate that HoneySat's radio simulator (which controls the communication between the ground and space segments) is considered to realistically simulate the communication of a real satellite mission by the majority of participants. A significant key finding of our survey is shown below. Overall, the results obtained provide convincing evidence for answering question Q-2 in the affirmative and design objective DO-2 as achieved.

**Key finding Q-2**

71.4% of surveyed satellite operators agreed or strongly agreed with the following statement: *"If I did not know this was a honeypot simulation of a CubeSat satellite mission, I would have believed it to be an actual CubeSat satellite mission."*

## 6.4 Internet Interaction Experiment

This experiment explores HoneySat's capabilities to entice external actors in the wild by deploying our honeypot over the Internet. To accomplish this, we leveraged the Shodan search engine. Shodan is a search engine for Internet-connected devices that scans the whole Internet and indexes exposed servers and their TCP ports. Shodan then reads the banner information for each port. For example, it gathers the web headers and Telnet login banners [40]. Additionally, Shodan tags servers as honeypots if the servers have too many open ports [37].

**Experiment Description.** We deployed and configured five HoneySat instances over the Internet and exposed TCP port 24 for the Telnet server and 80 for the web interface so that anybody can interact with it and allow Shodan to index our server and its banners. We deployed four HoneySat instances in total.

**Experiment Results.** Our honeypot servers were successfully indexed by Shodan. Both Telnet and web banners were captured, and none of them were tagged as honeypots.

| Honeypot | Source IP | Cmds Received | Engaged Time |
|---|---|---|---|
| Honeypot 1 | Egypt | 4 | 2 hr |
| Honeypot 2 | Sweden (Tor) | 9 | 5 min |
| Honeypot 1 | France (Tor) | 6 | 4 min |
| Honeypot 3 | USA | 8 | 3 min |

Table 5: Exposed Telnet interactions received.

Our honeypots successfully enticed external actors and captured 4 distinct sessions (show in Table 5) via the exposed Telnet protocol discussed in Sec. 4.3. The exposed Telnet protocol was implemented so that a human had to type *"activate"* in the terminal before sending commands. This feature was implemented to filter out Internet Telnet bots. As such, the commands that we describe below were sent by a human and not a crawler or Internet bot.

We now describe the commands received in the first interaction session shown in Table 5.

1. *help:* Shows the ground software available commands.
2. *ls:* The Linux ls command is not a valid command in the honeypot ground software.
3. *fp_show:* Prints the list of commands in the current flight plan (The flight plan refers to a pre-programmed sequence of tasks, and commands that guide the satellite's operation).
4. *com_debug:* Prints the current CSP network configuration including the current CSP node, interfaces and routing table.

The commands *fp_show* and *com_debug* are *flight software-specific commands* which suggests that the adversary was successfully deceived by our honeypot and purposefully sent flight software commands instead of haphazardly sending unrelated commands. Another factor that indicates that the adversary was deceived is that they engaged with our honeypot over a period of two hours indicating that they did not immediately identified it as a honeypot.

In summary, these results provide strong evidence for answering question Q-2 in the affirmative and design objective DO-2 as achieved.

**Key finding Q-2**

Human adversaries interacted with three of our HoneySat-powered honeypots during four distinct Telnet sessions resulting in *16 satellite mission-specific commands* captured.

## 6.5 Case Study: Generic CCSDS Mission Honeypot

In Sec. 5.2 we described how we implemented HoneySat using one SmallSat protocol ecosystem, namely, CSP. In this case study we are interested in testing the extensibility capabilities of HoneySat to support additional ecosystems (discussed in Sec. 2.5) by adding a *second ecosystem* to HoneySat, namely the CCSDS ecosystem.

**Experiment Description.** We selected the CCSDS ecosystem because it is a standard protocol suite used by other SmallSats [81]. To accomplish this, we leveraged an open-source CCSDS ecosystem-based flight software framework, RACCOON OS [31], and YAMCS, an open-source Mission Control software framework with built-in support for PUS [1].

**Methodology.** Building upon our HoneySat framework implementation, as detailed in Sec. 5, we enhanced the system by integrating various components of the RACCOON OS and the YAMCS framework.

**Results.** We successfully extended HoneySat to support the CCSDS ecosystem. Regarding the ground segment we implemented the *exposed network protocol* using YAMCS' built-in web interface (Fig. 7), for the *mission control software* we used YAMCS's built-in Mission Control Software [1], for the *radio simulator* we used RACCOON's communication application and for the *ground configuration* and *logging repository* we again used YAMCS built-in features.

Focusing on the space segment, we implemented the satellite *flight software runtime* using the RACCOON framework, and for the *Flight Software Services*, we configured the RACCOON framework to connect it to the YAMCS's MCS on the ground segment. The *satellite personality* and *logging repository* were based on the existing HoneySat implementations.

The majority of the effort involved in extending HoneySat to support the CCSDS ecosystem was dedicated to understanding the ecosystem itself, including components such as PUS. Additionally, we had to analyze the RACCOON flight software code and the YAMCS framework documentation. The only extra implementation that was required was the Flight Software Services which we modified using Rust. Other than that we reused several elements of HoneySat like the satellite personality and the Satellite Simulator.

In summary, these results provide strong evidence for answering question Q-3 in the affirmative and design objective DO-3 as achieved.

**Key findings Q-3**

Out of the box HoneySat supports CSP and CCSDS, the two most widely used standard space ecosystems, and was evaluated by simulating *3 real-world SmallSats*.

## 7 Discussion and Future Work

**Limitations.** Currently, the functionality of some subsystems within HoneySat's Satellite Simulator are constrained by the quality of the data provided. For example, the resolution of Earth's images generated by our camera payload depends on the resolution of its source data (USGS [76]). Consequently, creating a honeypot for a satellite with a high-resolution camera payload would not be feasible without addressing the underlying issues of data source quality.

**Broader Applications of HoneySat**. While originally designed as a honeypot, our framework offers the flexibility to support a range of applications beyond its initial purpose. One promising use case is the development of digital twins for satellite systems, enabling the simulation of real-world satellite subsystems and communication scenarios. Furthermore, HoneySat can be integrated into cyber range environments to enhance training programs and cybersecurity simulation exercises. Lastly, HoneySat serves as an educational tool, providing researchers, industry professionals, and security enthusiasts with an opportunity to explore and learn about satellite architecture and operations.

## 8 Conclusion

Although we have yet to witness a Stuxnet-like cyberattack on space systems, security researchers and professionals need to develop effective countermeasures to secure satellites. In this paper, we introduced HoneySat, the first satellite honeypot that provides a much-needed alternative source of empirical data on attackers' TTPs. We created a satellite honeypot that realistically simulates an entire satellite system, including its sensors and subsystems.

We performed several experiments that provide strong evidence that the framework can obtain rich interaction data, can be extended and customized, and simulate a satellite to keep its honeypot nature hidden from potential adversaries. Finally, we hope that security researchers and professionals take advantage of HoneySat's open-source implementation and use it as a foundation not only for satellite honeypot deployments but also for simulation, education, and training applications.

## 9 Ethical Considerations

In this paper we consider the ethical consequences and possible negative outcomes of two main elements of our research. The satellite operator user study and the multiple deployments of our satellite honeypot exposed to the Internet. We now discuss the stakeholders, potential risks and how we mitigated those risks for each of these elements.

### 9.1 Satellite Operators User Study

Our user study adheres to the ethical guidelines outlined in the USENIX Security '25 Ethics Policy. Prior to conducting any research, the study plan, survey protocol, and associated materials were submitted for review by our Institutional Review Board (IRB), which evaluated the study's potential risks to participants. The protocol received "exempt status" from the IRB, indicating that the research involves no more than minimal risk. This status reflects our careful evaluation of possible harms, which we discuss further below.

**Stakeholder Identification.** The primary stakeholders in the survey are *survey participants*. Our study aims to gather voluntary responses regarding the fidelity and realism of our honeypot system. We carefully considered the potential risks to both the participants of the survey.

**Risk Mitigation.**The main risk for the survey participants is the breach of privacy. To mitigate this risk, we ensured that no identifiable data (e.g., names, contact details) was collected from survey participants and the survey responses remain fully anonymous. Additionally, the survey itself includes an informed consent section that informs the respondents about the voluntary participation, the survey's purpose, a description of the procedures, the risks involved, and contact information and the option to opt out of the survey. Finally, we provided participants the choice to skip questions using the response options "Prefer not to say" and "I do not know" to applicable questions. Due to the aforementioned risk mitigation strategies the stakeholders' breach of privacy risk is negligible.

### 9.2 Honeypot Deployment Experiment

**Stakeholder Identification.** The primary stakeholders in the honeypot deployment are the *external actors* who interact with our honeypot and the *infrastructure owners* on whose infrastructure our honeypot is running. In the honeypot deployment experiment we deployed HoneySat on server infrastructure owned by real-world institutions.

**Risk Mitigation.** The main risk for the infrastructure owners is that one of the external actors will use our honeypot as an stepping stone to breach their production infrastructure. For the deployment on the premises of the institutions we worked with the local IT infrastructure administrators to ensure that we took all the necessary precautions to avoid any possibility that the external actors could gain access to the infrastructure via our honeypot. First, the infrastructure owners provided us with a server with a clean OS installation that did not have any production applications or sensitive data. Second, the server was isolated on its own network segment. Third, we configured a firewall to only allow the necessary ports. Fourth, the VNC portion of the honeypot is configured so that external actors cannot use it to host malicious services by denying traffic forwarding. Fifth and final, we deployed our honeypot using two sandboxing layers (Docker containers and Virtual Machines (VMs) as we discussed in Appendix A.3. Due to the aforementioned risk mitigation strategies the stakeholders' breach of privacy risk is negligible.

The main risk for the external actors is the breach of their privacy. However, it is generally accepted that trespassers of a computer system do not have reasonable expectation of privacy [66]. In order to mitigate this risk our honeypot system gives notice and warning to any user connecting to it indicating that "This computer system is for authorized use only." Additionally, the web services of our honeypot are protected by a username and password. This notice system gives external actors a chance to avoid interaction, which helps reduce the risk.

## References

[1] Yamcs Mission Control. `https://yamcs.org`, 2025. [Accessed 23-01-2025].

[2] Docker Inc. Docker: Accelerated container application development, April 2024.

[3] B. Acharya, M. Saad, A. Emanuele Cinà, L. Schönherr, H. Dai Nguyen, A. Oest, P. Vadrevu, and T. Holz. Conning the crypto conman: End-to-end analysis of cryptocurrency-based technical support scams. In *2024 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[4] European Space Agency. ESA SPACE-SHIELD, 2023.

[5] Robin Bisping, Johannes Willbold, Martin Strohmeier, and Vincent Lenders. Wireless signal injection attacks on VSAT satellite modems. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6075–6091, Philadelphia, PA, August 2024. USENIX Association.

[6] Robert Bocchino, Timothy Canham, Garth Watney, Leonard Reder, and Jeffrey Levison. F prime: An open-source framework for small-scale flight software systems. In *AIAA/USU Conference on Small Satellites, Advanced Technologies II, SSC-18-XII-04*, Logan, UT, 2018.

[7] Nicolò Boschetti, Nathaniel G Gordon, and Gregory Falco. Space cybersecurity lessons learned from the viasat cyberattack. In *ASCEND 2022*, page 4380. 2022.

[8] Matt Burgess. A clever honeypot tricked hackers into revealing their secrets, August 2023.

[9] John Burkardt. Rk4 - runge-kutta 4th order ode solver, October 2016.

[10] San Luis Obispo California Polytechnic State University. Earth station - polysat, April 2024.

[11] Van Camp Charlotte and Peeters Walter. A world without satellite data as a result of a global cyber-attack. *Space Policy*, 59:101458, 2022.

[12] Fred Cohen. The use of deception techniques: Honeypots and decoys. *Handbook of Information Security*, 3(1):646–655, 2006.

[13] Frederick Cohen. Deception toolkit, March 1998.

[14] Mauro Conti, Francesco Trolese, and Federico Turrin. Icspot: A high-interaction honeypot for industrial control systems. In *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–4, 2022.

[15] Alexandru Csete. Gpredict: Free, real-time satellite tracking and orbit prediction software, December 2023.

[16] Jorg Daubert, Dhanasekar Boopalan, Max Mühlhäuser, and Emmanouil Vasilomanolakis. Honeydrone: A medium-interaction unmanned aerial vehicle honeypot. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018.

[17] DLR. Open modUlar sofTware PlatfOrm for SpacecrafT, 2022.

[18] Jakub Drmola and Tomas Hubik. Kessler syndrome: System dynamics model. *Space Policy*, 44:29–39, 2018.

[19] European Union Agency for the Space Programme. Copernicus | eu agency for the space programme, April 2024.

[20] Lorenzo Franceschi-Bicchierai. Thousands of new honeypots deployed across israel to catch hackers, November 2023.

[21] Cristobal Garrido, Elias Obreque, Matias Vidal-Valladares, Samuel Gutierrez, and Marcos Diaz Quezada. The first chilean satellite swarm: Approach and lessons learned. In *AIAA/USU Conference on Small Satellites, Year in Review - Research & Academia, SSC23-WVII-07*, Logan, UT, 2023.

[22] Steven B Giddings. Hawking radiation, the stefan–boltzmann law, and unitarization. *Physics Letters B*, 754:39–42, 2016.

[23] GomSpace. Nanocom ms100 datasheet, March 2021.

[24] Carlos E. Gonzalez, Camilo J. Rojas, Alexandre Bergel, and Marcos A. Diaz. An architecture-tracking approach to evaluate a modular and extensible flight software for cubesat nanosatellites. *IEEE Access*, 7:126409–126429, 2019.

[25] Brian E. Granger and Min Ragan-Kelley. Pyzmq documentation, April 2024.

[26] Jose Angel Gutierrez Ahumada, Kelsey Doerksen, and Stefan Zeller. Automated fleet commissioning workflows at planet. In *AIAA/USU Conference on Small Satellites, Technical Session 12: Constellation Missions, SSC1-XII-04*, Logan, UT, 2021.

[27] Stephen Hilt, Federico Maggi, Charles Perine, Lord Remorin, Martin Rösler, and Rainer Vosseler. Caught in the act: Running a realistic factory honeypot to capture real threats. *Trend Micro Research*, 2020.

[28] Mark Holmes. The growing risk of a major satellite cyber attack, May 2023.

[29] Thorsten Holz and Frederic Raynal. Detecting honeypots and other suspicious environments. In *Proceedings from the sixth annual IEEE SMC information assurance workshop*, pages 29–36, West Point, NY, USA, 2005. IEEE.

[30] Niclas Ilg, Paul Duplys, Dominik Sisejkovic, and Michael Menth. A survey of contemporary open-source honeypots, frameworks, and tools. *Journal of Network and Computer Applications*, 220:103737, 2023.

[31] Phillip Wüstenberg José Manual Diez, Fabian Krech. Raccoon os. https://gitlab.com/rccn. [Accessed 20-01-2025].

[32] JSON. Json, April 2024.

[33] Georgios Kavallieratos and Sokratis Katsikas. An exploratory analysis of the last frontier: A systematic literature review of cybersecurity in space. *International Journal of Critical Infrastructure Protection*, page 100640, 2023.

[34] Erik Kulu. SUCHAI 2 @ Nanosats Database — nanosats.eu. https://www.nanosats.eu/sat/suchai-2, 2022. [Accessed 29-04-2024].

[35] Efrén López-Morales. Securing cyber-physical systems via advanced cyber threat intelligence methods. CCS '24, page 5119–5121, New York, NY, USA, 2024. Association for Computing Machinery.

[36] Efrén López-Morales, Ulysse Planta, Carlos Rubio-Medrano, Ali Abbasi, and Alvaro A. Cardenas. SoK: Security of programmable logic controllers. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7103–7122, Philadelphia, PA, August 2024. USENIX Association.

[37] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 279–291, New York, NY, USA, 2020. Association for Computing Machinery.

[38] Marco Lucchese, Francesco Lupia, Massimo Merro, Federica Paci, Nicola Zannone, and Angelo Furfaro. Honeyics: A high-interaction physics-aware honeynet for industrial control systems. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.

[39] Gabriele Marra, Ulysse Planta, Philipp Wüstenberg, and Ali Abbasi. On the feasibility of cubesats application sandboxing for space missions. In *Workshop on the Security of Space and Satellite Systems (SpaceSec)*, 2024.

[40] John Matherly. Complete guide to shodan. *Shodan, LLC (2016-02-25)*, 1, 2015.

[41] David McComas, Jonathan Wilmot, and Alan Cudmore. The core flight system (cfs) community: Providing low cost solutions for small spacecraft. In *Annual AIAA/USU Conference on Small Satellites*, Logan, UT, 2016. Utah State University, University Libraries.

[42] Mario Merri, Alessandro Ercolani, Damiano Guerrucci, Vemund Reggestad, and David Verrier. Cutting the cost of esa mission ground software, May 2007.

[43] MongoDB, Inc. Mongodb: The developer data platform | mongodb, April 2024.

[44] NASA. What are smallsats and cubesats?

[45] NASA. What is remote sensing? | earthdata, August 2019.

[46] NASA Goddard Space Flight Center. Flight training: Introduction.

[47] National Institute of Standards and Technology (NIST). tactics, techniques, and procedures (ttp) - glossary, April 2024.

[48] Jose Nazario. paralax/awesome-honeypots: an awesome list of honeypot resources, March 2024.

[49] Nmap. Os detection, January 2025.

[50] Michel Oosterhof. cowrie/cowrie: Cowrie ssh/telnet honeypot https://cowrie.readthedocs.io, April 2024.

[51] Pierre Ossman. Tigervnc/tigervnc: High performance, multi-platform vnc client and server, July 2024.

[52] James Pavur and Ivan Martinovic. Building a launchpad for satellite cyber-security research: lessons from 60 years of spaceflight. *Journal of Cybersecurity*, 8(1):tyac008, 2022.

[53] Ulysse Planta, Julian Rederlechner, Gabriele Marra, and Ali Abbasi. Let me do it for you: On the feasibility of inter-satellite friendly jamming. In *2024 Security for Space Systems (3S)*, pages 1–6, 2024.

[54] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, Boston, MA, USA, 2007.

[55] Qualtrics. What is a likert scale?, January 2025.

[56] Muhammad Raza. What are ttps? tactics, techniques & procedures explained, April 2024.

[57] Brandon Rhodes. skyfielders/python-skyfield: Elegant astronomy for python, April 2024.

[58] Luis Salazar, Efrén López-Morales, Juan Lozano, Carlos Rubio-Medrano, and Álvaro A. Cárdenas. Icsnet: A hybrid-interaction honeynet for industrial control systems. In *Proceedings of the Sixth Workshop on CPS&IoT Security and Privacy*, CPSIoTSec'24, page 68–79, New York, NY, USA, 2024. Association for Computing Machinery.

[59] Mario Merri Sam Cooper. Ccsds mission operations. https://indico.esa.int/event/62/contributions/2797/attachments/2307/2667/1235_-_mission-operation-services---future-trends_Presentation.pdf. [Accessed 20-01-2025].

[60] Tobias Scharnowski, Felix Buchmann, Simon Wörner, and Thorsten Holz. A case study on fuzzing satellite firmware. In *Workshop on the Security of Space and Satellite Systems (SpaceSec)*, 2023.

[61] Mitch Semanik and Patrick Crotty. U.s. private space launch industry is out of this world, November 2023.

[62] Shared Threat Intelligence for Network Gatekeeping and Automated Response (STINGAR). About - stingar, April 2024.

[63] Yasushi Shoji. libcsp/libcsp, February 2024.

[64] Yasushi Shoji. The Protocol Stack — Cubesat Space Protocol, 2024.

[65] Space and Planetary Exploration Laboratory, University of Chile. SPEL / SUCHAI-Flight-Software · GitLab, February 2024.

[66] Lance Spitzner. *Honeypots: tracking hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[67] Robert L Staehle, Brian Anderson, Bruce Betts, Diana Blaney, Channing Chow, Louis Friedman, Hamid Hemmati, Dayton Jones, Andrew Klesh, Paulett Liewer, et al. Interplanetary cubesats: opening the solar system to a broad community at lower cost. Technical report, NASA, 2012.

[68] The Aerospace Corporation. Sparta, April 2024.

[69] The European Space Agency. Esa - telemetry & telecommand, March 2013.

[70] The European Space Agency. Esa - about payload systems, April 2024.

[71] The MITRE Corporation. Groups | mitre att&ck, April 2024.

[72] The ZeroMQ authors. ZeroMQ | get started, 2024.

[73] Clifford A Truesdell. *A First Course in Rational Continuum Mechanics V1*. Academic Press, 1992.

[74] United States Space Force. Global positioning system > space operations command (spoc) > display, February 2023.

[75] U.S. Geological Survey. Uncrewed aerial systems | u.s. geological survey, February 2004.

[76] U.S. Geological Survey. Earthexplorer | u.s. geological survey, November 2022.

[77] David A Vallado and Paul J Cefola. Two-line element sets–practice and use. In *63rd International Astronautical Congress*, pages 1–14, Naples, Italy, 2012. International Astronautical Federation.

[78] Johnny Vestergaard. mushorg/conpot: Ics/scada honeypot, March 2024.

[79] Johannes Willbold, Tobias Cloosters, Simon Wörner, Felix Buchmann, Moritz Schloegel, Lucas Davi, and Thorsten Holz. Space RadSim: Binary-Agnostic Fault Injection to Evaluate Cosmic Radiation Impact on Exploit Mitigation Techniques in Space . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1047–1063, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[80] Johannes Willbold, Moritz Schloegel, Robin Bisping, Martin Strohmeier, Thorsten Holz, and Vincent Lenders. Vsaster: Uncovering inherent security issues in current vsat system practices. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 288–299, 2024.

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

[82] Lloyd Wood. Introduction to satellite constellations: orbital types, uses and related facts, July 2006.

[83] Xplore, Inc. kubos/kubos, February 2024. original-date: 2018-01-18T14:55:14Z.

[84] Bruce D. Yost. Nasa ssri knowledge base | detailed design and analysis > subsystem design > command and data handling, June 2023.

## A Appendix

### A.1 Interaction Sequences and Exploits

Table 6 includes all the interaction sequences and exploits we performed during the experiments in Sec. 6.2.

### A.2 Satellite Simulator Design Details

As shown in Fig. 5, the Satellite Simulator includes five module types: *simulation modules* (light orange), *sensor modules* (light blue), *subsystem state modules* (light green), *satellite state modules* (light gray) and *interface modules* (light yellow).

*Simulation Modules*. These modules are abstractions of real-world processes required for a realistic satellite simulation (DP-2). An example of these simulations is the orbital simulation. This simulation uses orbital mechanics to calculate data such as the satellite's orbital position at any given time.

These simulation modules can communicate with each other to facilitate proper functionality. For example, the power simulation module will query the orbital simulation for orbital data to determine if the satellite is positioned properly in relation to the sun so that it can draw power from its solar panels. The subsystems and the sensors of the Satellite Simulator are designed around these simulations. The Satellite Simulator includes the orbital, rotation, power system, thermal, magnetic, and payload simulations. Due to space restrictions and their complex design, we explain each simulation in Appendix A.2.

Overall, these simulations provide the foundation for any satellite operator to use our framework and implement their

Table 6: Tactics, Techniques and Procedures' Interaction Experimental Exploits.

| Tactic | Technique | ID | Subsystem | Exploit |
|---|---|---|---|---|
| Reconnaissance | Active Scanning (RF/Optical) | T2001 | Threat Model Limitation | N.A. |
| Reconnaissance | Gather Victim Mission Information | T2002 | Web Interface | Use the Web Interface to gather mission documentation. |
| Reconnaissance | Gather Victim Org Information | T1591 | Web interface | Use the Web Interface to gather mission documentation. |
| Reconnaissance | In orbit proximity intelligence | T2029 | Threat Model Limitation | N.A. |
| Reconnaissance | Passive Interception (RF/Optical) | T2004 | Threat Model Limitation | N.A. |
| Reconnaissance | Phishing for Information | T1598 | Tangential | N.A. |
| Resource Development | Acquire or Build Infrastructure | T1583 | Telnet interface | Acquire ground segment using the Telnet service. |
| Resource Development | Compromise Account | T2038 | Tangential | N.A. |
| Resource Development | Compromise Infrastructure | T1584 | Threat Model Limitation | N.A. |
| Resource Development | Develop/Obtain Capabilities | T2007 | Ground software. Flight software. | Exploit OS, libraries or software vulnarabilities. Deploy custom CSP application to forge TC/TM. |
| Initial Access | Direct Attack to Space Communication Links | T2008 | Ground software. Flight software. | Use the ground software to send/receive TC/TM. Deploy custom CSP application to forge TC/TM. |
| Initial Access | Ground Segment Compromise | T2030 | Telnet interface. Ground software. | Use the telnet interface to access the ground software. |
| Initial Access | Supply Chain Compromise | T1195 | Tangential | N.A. |
| Initial Access | Trusted Relationship | T2039 | Threat Model Limitation | N.A. |
| Initial Access | Valid Credentials | T2009 | Tangential | N.A. |
| Execution | Modification of On Board Control Procedures modification | T2010 | Ground software. Flight software. | Upload a script to the satellite software: tm_send_file code.py 1 1: obc_system python recv_files/code.py |
| Execution | Native API | T1106 | Ground software. Flight software. | Execute shell commands or delete system files: 1: obc_system <command> 1: obc_rm -r $HOME |
| Execution | Payload Exploitation to Execute Commands | T2012 | Tangential | N.A. |
| Persistence | Backdoor Installation | T2014 | Ground software. Flight software. | Upload a script to the satellite software: tm_send_file backdoor.sh 1 1: obc_system ./recv_files/backdoor.sh |
| Persistence | Key Management Infrastructure Manipulation | T2013 | Tangential | N.A. |
| Persistence | Pre-OS Boot | T1542 | Ground software. Flight software. | Use obc_system, obc_rm, obc_mkdir commands. Upload/modify an OS configuration file. Start/stop/schedule execution of services/daemons. |
| Persistence | Valid Credentials | T2009 | Tangential | N.A. |
| Privilege Escalation | Become Avionics Bus Master | T2031 | Implementation limitation | N.A. |
| Privilege Escalation | Escape to Host | T1611 | Docker. Virtual machine. | Escape the container/VM with previously crafted exploits. |
| Defense Evasion | Impair Defenses | T1562 | Ground software. Flight software. | Send commands to change operation mode. 1: drp_set_var_name obc_opmode 0 |
| Defense Evasion | Indicator Removal on Host | T1070 | Ground software. Flight software. | Use commands to remove artifacts, logs, etc.: 1: obc_rm <path> |
| Defense Evasion | Masquerading | T2040 | Ground software. Flight software. | Use commands to upload artifacts modify system settings: tm_send_file articaft 1 1: obc_mv artifact /etc/config/artifact |
| Defense Evasion | Pre-OS Boot | T2041 | Ground software. Flight software. | Use obc_system, obc_rm, obc_mkdir commands. Upload/modify an OS configuration file. Start/stop/schedule execution of services/daemons. |
| Credential Access | Adversary in the Middle | T2042 | Ground software. Flight software. | Deploy a CSP application to capture/inject CSP packets. |
| Credential Access | Brute Force | T2043 | Ground software. Flight software. | Brute force valid TC parameters: 1: obc_ebf <KEY> |
| Credential Access | Communication Link Sniffing | T2044 | Ground software. | Scape the ground software or docker and run tcpdump. Deploy a CSP application to capture/inject CSP packets. |
| Credential Access | Retrieve TT&C master/session keys | T2015 | Tangential | N.A. |
| Discovery | Key Management Policy Discovery | T2032 | Tangential | N.A. |
| Discovery | Spacecraft's Components Discovery | T2034 | Ground software. Flight software. | Send TC to redirect satellite logs to ground segment: 1: log_set 5 2 10 |
| Discovery | System Service Discovery | T1007 | Ground software. Flight software. | Capture running processes information 1: obc_system ps -aux >ps.log 1: tm_send_file 10 ps.log |
| Discovery | Trust Relationships Discovery | T2033 | Tangential | N.A. |
| Lateral Movement | Compromise a Payload after compromising the main satellite platform | T2045 | Implementation Limitation | N.A. |
| Lateral Movement | Compromise of satellite hypervisors | T2017 | Docker. Virtual machine. | Escape the container/VM with previously crafted exploits. |
| Lateral Movement | Compromise the satellite platform starting from a compromised payload. | T2046 | Implementation Limitation | N.A. |
| Lateral Movement | Lateral Movement via common Avionics Bus. | T2016 | Implementation Limitation | N.A. |
| Collection | Adversary in the Middle | T1557 | Ground software. Flight software. | Deploy a CSP application to capture/inject CSP packets. |
| Collection | Data from link eavesdropping | T2018 | Ground software. Flight software. | Scape the ground software or docker and run tcpdump. Deploy a CSP application to capture/inject CSP packets. |
| Command and Control | Protocol Tunnelling | T2047 | Ground software. Flight software. | Deploy an malicious application that sends data over CSP |
| Command and Control | Telecommand a Spacescraft | T2019 | Ground software. Flight software. | Use the ground software to send TC: 1: com_ping 1 |
| Command and Control | TT&C over ISL | T2048 | Threat Model Limitation | N.A. |
| Exfiltration | Exfiltration Over Payload Channel | T2021 | Implementation Limitation | N.A. |
| Exfiltration | Exfiltration Over TM Channel | T2022 | Ground software. Flight software. | The atacker deploys a custom CSP node or a backdoor |
| Exfiltration | Optical link modification | T2037 | Threat Model Limitation | N.A. |
| Exfiltration | RF modification | T2036 | Threat Model Limitation | N.A. |
| Exfiltration | Side-channel exfiltration | T2035 | Threat Model Limitation | N.A. |
| Impact | Data Manipulation | T2054 | Ground software. Flight software. | Send TC to modify/reset TM database: 1: drp_set_var_name drp_ack_ads 10000000 1: drp_reset_payload 1 1010 1: drp_reset_status 1010 |
| Impact | Ground Segment Jamming | T2050 | Threat Model Limitation | N.A. |
| Impact | Loss of spacecraft telecommanding | T2055 | Ground software. Flight software. | Send TC to change communication paramters. Modify network configuration in the ground station. |
| Impact | Permanent loss to telecommand satellite | T2027 | Ground software. Flight software. | Send TC to destroy filesystem: 1: obc_system rm -rf –no-preserve-root / |
| Impact | Resource damage | T2028 | Threat Model Limitation | N.A. |
| Impact | Resource Hijacking | T1496 | Ground software. Flight software. | Upload a script to the satellite software: tm_send_file code.py 1 1: obc_system python recv_files/code.py |
| Impact | Saturation of Inter Satellite Links | T2052 | Threat Model Limitation | N.A. |
| Impact | Saturation/Exhaustion of Spacecraft Resources | T2053 | Ground software. Flight software. | Send TC to create a reset loop: 1: fp_set_cmd_dt 10 2147483647 10 obc_reset |
| Impact | Service Stop | T1489 | Ground software. Flight software. | Send TC to lauch a fork bomb or a reset loop 1: obc_system :(){ :\|:& };: 1: fp_set_cmd_dt 10 2147483647 10 obc_reset |
| Impact | Spacecraft Jamming | T2049 | Threat Model Limitation | N.A. |
| Impact | Temporary loss to telecommand satellite | T2026 | Ground software. Flight software. | Send TC to make the system unresponsive 1: obc_system sleep 3600 |
| Impact | Transmitted Data Manipulation | T2024 | Threat Model Limitation | N.A. |

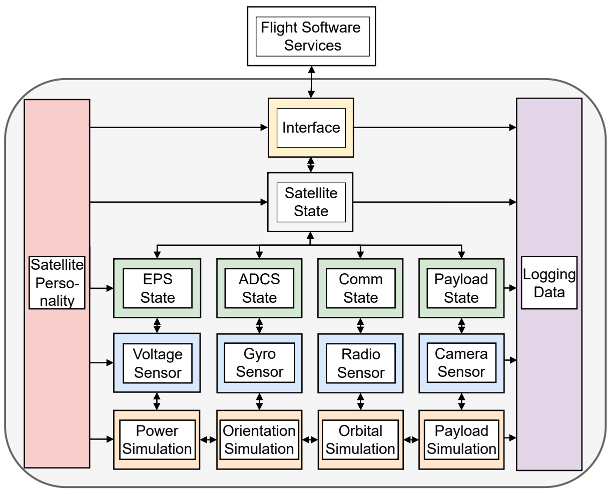


Figure 5: High-level architecture of the Satellite Simulator. Due to space limitations, we do not show all the available sensors and simulations. Some sensors communicate with multiple simulations. Although the flight software services are not part of the Satellite Simulator we included it to depict the Interface module's data flow better.

honeypot using their satellite flight software. The Satellite Simulator simulations are configured using the *satellite personality*, which we discuss below.

*Sensor Modules.* These modules are abstractions of the hardware sensors used by a satellite, i.e., temperature sensor. The sensor modules do not perform any computations, instead they collect data directly from the simulation modules following DP-2. For example, the voltage sensor will query the power simulation to collect data about the current state of the battery.

*Subsystem State Modules.* These modules are abstractions of the satellite subsystems discussed in Sec. 2.4. A given subsystem state is made up of one or more sensors. For example, in Fig. 5 the *Payload State* (light green) includes the *Camera Sensor* (light blue). In this way, subsystem states serve two purposes. They can *get* data from their sensors or can *set* a specific configuration on their sensors.

*Satellite State Module.* This module is an abstraction of an entire satellite. As depicted in Fig. 5 the *Satellite State* (light gray) is made up of multiple subsystem states. Satellite State module routes messages between the Satellite Simulator and the modified flight software. For example, if the modified flight software sends a message to the Satellite Simulator requesting to provide the present voltage in the EPS, the satellite state will route that message to the EPS State.

*Interface Module.* This module is an abstraction of the communication protocol between the modified flight software and the Satellite Simulator. As depicted in Fig. 5, the *Interface* (light yellow) is the module that connects the flight software services with the Satellite Simulator simulations.

This protocol must implement two basic message types, *requests* and *replies*. However, to meet DO-3, the underlying implementation of these messages is left open for the user to decide based on their requirements.

To finish, the Satellite Simulator's built-in modular design allows subsystems, simulations, and sensor modules to be *mixed 'n matched* to meet the requirements of most satellites and their missions. Additionally, our flexible design allows users to easily add additional simulations, sensors, or subsystems as modules to the honeypot.

- *Orbital Simulation.* This simulation calculates and predicts the satellite's orbit. To achieve this, it uses the two-line element set or TLE data configured in the satellite personality. The TLE data contains the necessary orbital information required to calculate the location of a satellite given a point in time [77]. We rely on the Skyfield python library to perform advanced and precise calculations [57]. This simulation calculates multiple orbital values which include the altitude angle, latitude, longitude of the satellite among other values. The payload, EPS, and magnetic simulations rely on data from this simulation.

- *Rotation Simulation.* This simulation simulates the satellite attitude changes to provide the satellite orientation. Determining and controlling the satellite orientation is relevant for payloads such as cameras or communication systems that require pointing to the earth surface or to other celestial bodies. This simulation calculates the rotation kinematics of the satellite using two reference frames: a reference frame fixed to the satellite body and a non-rotational reference frame. The relation between the two reference frames is calculated using the conservation of angular momentum and the rigid-body Euler equation [73]. To simulate the satellite's rotation, the initial instantaneous angular velocity of the satellite, the current orientation in quaternions, and the satellite's inertia matrix must be defined. The simulation uses the fourth-order Runge Kutta integration method (RK4) [9] to perform these calculations to avoid divergences in the simulation with time steps greater than 1 second.

- *Power System Simulation.* This simulation manages the satellite's power collection, consumption, and distribution. It tracks the battery capacity, the power draw and simulates battery charging whenever the orbital simulation tells it that the satellite is exposed to sunlight. A battery pack is modeled as a series or parallel configuration of smaller battery packs. The simplest battery pack consists of a single cell. A single cell is simulated to calculate the charge and discharge curves, assuming that the cell's current will not change. This cell voltage is then used to calculate the pack's voltage. The electrical power system simulation uses the pack's voltage together with the input and output power

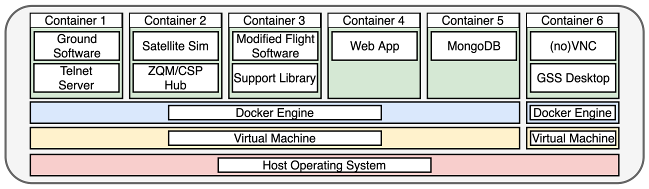


Figure 6: Architecture of dockerized Generic CSP Honeypot

to calculate the battery current. In case the input or output power of the power subsystem changes, a new cell current is calculated, and the battery cell simulation is rerun to reflect the change.

- *Thermal Simulation.* This simulation calculates the temperature of the satellite. The simulation is based on two values: the total thermal energy and a user-defined specific heat capacity. Together with emissivity, mass and surface area, the temperature of a satellite can be approximated based on the absorbed sunlight and the energy emitted through thermal radiation. This simulation uses a radiation loss formula [22] to calculate the thermal radiation emission.

- *Magnetic Simulation.* This simulation tracks and analyzes the interactions between the Earth's magnetic field and the satellite's own magnetic environment. It communicates with the orbital and rotation simulations to determine the satellite's position and orientation in space so that it can calculate the Earth's magnetic field components (north, east, vertical) and total intensity at the satellite's current location.

- *RGB Camera Simulation.* This simulation replicates the functionality of an RGB camera pointed at Earth. It uses Earth observation satellite imagery from the EarthExplorer database from the U.S. Geological Survey (USGS) [76]. If a capture image command is issued, it will select an image from one of two pools of images: daytime or nighttime.

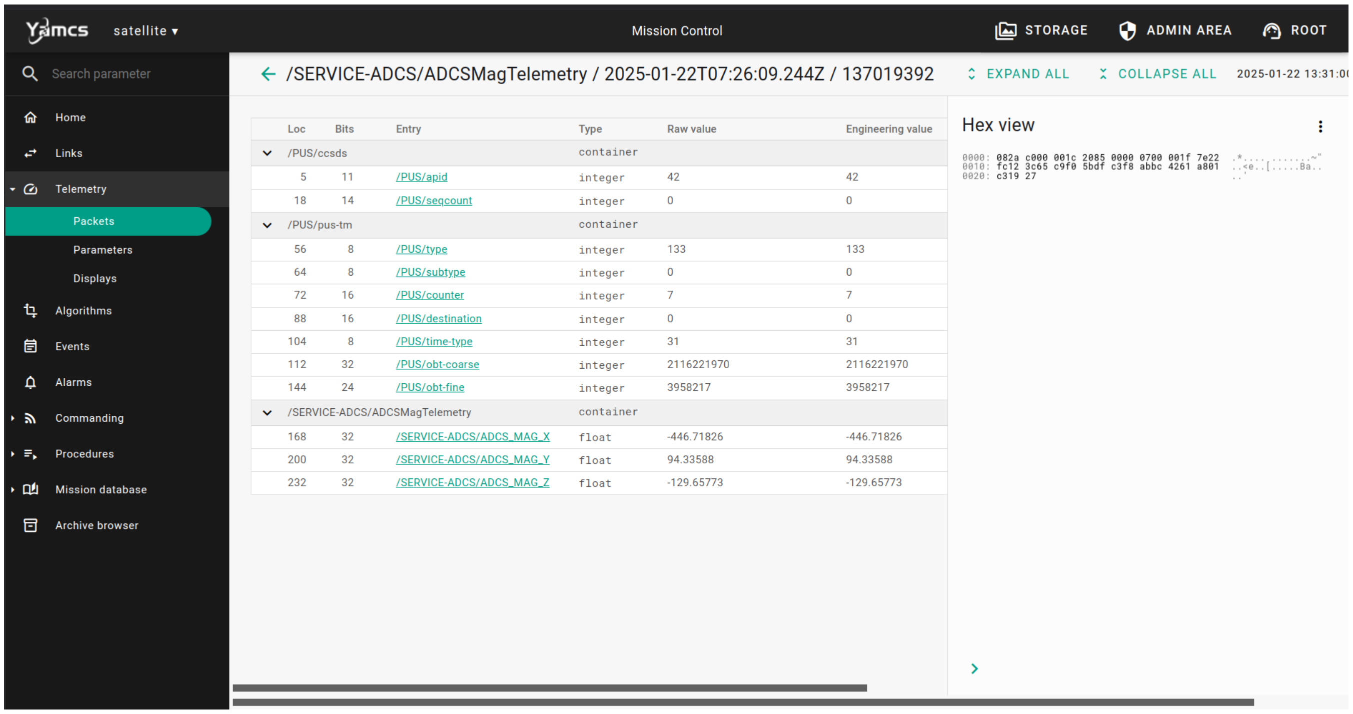
  This simulation communicates with the orbital simulation to determine if satellite is in the presence of the sun.

## A.3 Deployment and Security Hardening Implementation

As we mentioned in Sec. 2.1, high-interaction honeypots such as HoneySat present a high risk of adversary takeover. To mitigate this risk, we implemented HoneySat with two sandboxing layers.

**Virtual Machine.** VMs provide the highest isolation level among sandboxing techniques. We implemented HoneySat in VM environments to leverage VMs' robust security.

**Containerization.** After we completed HoneySat's development, we used Docker Compose [2] to containerize each of our framework's applications. Specifically, we created four different containers depicted in Fig. 6. Containerizing HoneySat provides two benefits. First, it creates another sandboxing layer that prevents adversaries from using our honeypot to access the underlying system [39]. Second, it proves a convenient and flexible way to deploy HoneySat.



/SERVICE-ADCS/ADCSMagTelemetry / 2025-01-22T07:26:09.244Z / 137019392

2025-01-22 13:31:0

| Loc | Bits | Entry | Type | Raw value | Engineering value |
|---|---|---|---|---|---|
| | | /PUS/ccsds | container | | |
| 5 | 11 | /PUS/apid | integer | 42 | 42 |
| 18 | 14 | /PUS/seqcount | integer | 0 | 0 |
| | | /PUS/pus-tm | container | | |
| 56 | 8 | /PUS/type | integer | 133 | 133 |
| 64 | 8 | /PUS/subtype | integer | 0 | 0 |
| 72 | 16 | /PUS/counter | integer | 7 | 7 |
| 88 | 16 | /PUS/destination | integer | 0 | 0 |
| 104 | 8 | /PUS/time-type | integer | 31 | 31 |
| 112 | 32 | /PUS/obt-coarse | integer | 2116221970 | 2116221970 |
| 144 | 24 | /PUS/obt-fine | integer | 3958217 | 3958217 |
| | | /SERVICE-ADCS/ADCSMagTelemetry | container | | |
| 168 | 32 | /SERVICE-ADCS/ADCS_MAG_X | float | -446.71826 | -446.71826 |
| 200 | 32 | /SERVICE-ADCS/ADCS_MAG_Y | float | 94.33588 | 94.33588 |
| 232 | 32 | /SERVICE-ADCS/ADCS_MAG_Z | float | -129.65773 | -129.65773 |

Hex view

```
0000: 082a c000 001c 2085 0000 0700 001f 7e22  .*.............~"
0010: fc12 3c65 c9f0 5bdf c3f8 abbc 4261 a801  ..<e..[.....Ba..
0020: c319 27                                  ..'
```

Figure 7: Screenshot of the YAMCS web interface, displaying a TM packet, returned from the Generic CCSDS Mission Honeypot.