Artifact Evaluated NDSS
Available
Functional
Reproduced

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[†]
[*]Equal contribution, joint first authors
[‡]New Mexico State University [†]CISPA Helmholtz Center for Information Security
[‖]Texas A&M University–Corpus Christi [§]Universidad de Santiago de Chile
[¶]Universidad de Chile [††]German Aerospace Center (DLR), Braunschweig, Germany

***Abstract*—Satellites are the backbone of mission-critical services that enable our modern society to function, for example, GPS. For 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. Unfortunately, there is no way to collect data on satellite adversarial techniques, hindering the generation of intelligence that leads to the development of countermeasures.**

In this paper, we present HoneySat, the first high-interaction satellite honeypot framework, capable of convincingly simulating a real-world CubeSat, a type of Small Satellite (SmallSat). To provide evidence of HoneySat's effectiveness, we surveyed SmallSat operators and deployed HoneySat over the Internet.

Our results show that 90% of satellite operators agreed that HoneySat provides a realistic simulation. Additionally, HoneySat successfully deceived adversaries in the wild and collected 22 real-world adversarial interactions. Finally, we performed a hardware-in-the-loop operation where HoneySat successfully communicated with an in-orbit, operational SmallSat mission.

I. Introduction

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 [1]. In addition, spacecraft can range from being as massive as thousands of kilograms, such as the International Space Station (ISS), to much smaller one-kilogram *CubeSats* [2]. As a result, the software and hardware components that make up a specific spacecraft vary greatly. 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 [3]. It could also cause food shortages and the freezing of financial transactions [4]. Furthermore, such an attack could exacerbate the Kessler Syndrome [5], 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 [3]: 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 [6]. Second, ground station technology has become significantly more affordable (and sometimes open source), greatly lowering the communication barrier with satellites [7]. 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 [8].

Although satellite security research has gained increased attention [8], [9], the relationship between outer space and cyberspace remains poorly understood [10]. Although the volume of reported cyber incidents in the space sector has grown, these reports rarely provide sufficient detail [4]. As a result, the security community has limited visibility into adversarial activity aimed at space infrastructure.

A honeypot's ability to collect real-world cyberattack data makes it an ideal solution to this problem [11]. 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 interactions. These logs can be analyzed to discover Tactics Techniques and Procedures (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 [14], [15], [16], [17], [18]. Honeypots are also used to deter malicious actors from attacking different types of systems, from industrial control systems [18] to social media platforms [19]. However, as of the time of writing, *there is no space-sector specific honeypot* in the literature.

In this paper, we design, implement, and evaluate the first satellite honeypot, *HoneySat*, to attract and analyze adversaries who attack space infrastructures over the Internet, a commonly observed threat vector [20], [21], [22], [23].

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240537
www.ndss-symposium.org

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 [24]. 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.

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 simulates an entire satellite mission, provides realistic telemetry, and supports real telecommands.

Finally, we collaborated with an aerospace company to perform a hardware-in-the-loop experiment where HoneySat successfully communicated with an in-orbit, operational SmallSat.

In summary, this paper makes the following contributions:

- We introduce HoneySat, a novel, high-interaction, extensible honeypot framework for small satellites (Sec. IV).
- We present the Satellite Simulator, a library that simulates the physical processes, sensors, and subsystems necessary for a realistic satellite honeypot (Sec. V).
- We demonstrate HoneySat's high level of realism through a hands-on survey where 10 experienced satellite operators interacted with HoneySat in *real time* (Sec. VI-C).
- We show HoneySat's extensibility features by integrating two real-world satellite flight software and integrating HoneySat into a hardware-in-the-loop operation allowing our honeypot to communicate with an in-orbit operational SmallSat (Sec. VI-E and Sec. VI-F).

In the spirit of open and reproducible science, HoneySat's source code and experimental results are available online[1],[2].

II. Background

This section lays out key background concepts that are relevant to satellite honeypots. For honeypots: a description of the different existing types (Sec. II-A), as well as the current state-of-the-art (Sec. II-B). For satellites: their operation (Sec. II-C), architecture (Sec. II-D), existing protocol ecosystems (Sec. II-E), and tactics, techniques, and procedures (TTPs) (Sec. II-F).

A. 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. They are easy to setup and maintain, and have a reduced risk of adversarial takeover. However, they provide limited interaction opportunities, which limits the data they provide. Some examples include Conpot [25] and Honeyd [26].

[1]https://doi.org/10.5281/zenodo.17871431

[2]https://github.com/HoneySat

TABLE I
Comparison of Existing Honeypots and HoneySat.

Keys: ✓ = Supported; ✗ = Not Supported.

Honeypot / Feature	Interaction Level	Included Protocols	Physics Sims	Extensibility
Conpot [25]	Low	9	0	✓
HoneyPLC[18]	High	3	0	✓
ICSPot [33]	High	4	1	✗
HoneyICS[31]	High	2	1	✓
HoneyDrone [32]	Medium	4	1	✗
HoneySat	High	4	6	✓
Addressed in Section	IV-D, IV-C	IV-C, V-B1	IV-D, V-A	VI-E VI-F

High-Interaction Honeypots. These honeypots offer extensive interaction opportunities via emulation or simulation [26]. Their main advantage is providing adversaries with almost limitless interactions. However, they pose a high takeover risk [27]. Examples include Cowrie [28] and HoneyPLC [18].

B. Honeypot's State of the Art

The literature on honeypots includes hundreds of implementations that simulate a diverse set of systems [29], [30]. From implementations that simulate a host's TCP/IP stack, such as Honeyd [26], to modern approaches that integrate social media applications, such as HoneyTweet [19]. In the absence of a satellite honeypot, we examine the approaches most related to satellites: Industrial Control Systems (ICS) and Unmanned Aerial Vehicles (UAV), summarized in Table I.

Satellite systems like ICS must be aware of some physical process, e.g., the sun's position, via *sensors* to acquire data about the physical world. ICS honeypots, such as HoneyICS [31], simulate these physical processes. In addition to ICS, UAV honeypots such as HoneyDrone [32] achieve this via simulations, and recreate attack scenarios.

C. Anatomy of a Satellite Mission

We now describe the components of a satellite mission. Due to satellite missions' complexity and diversity, 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 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, and the systems to operate the satellites, e.g., servers [8]. The GS includes the Ground Segment Software (GSS) that helps operators schedule and send commands and visualize data.

Space Segment ②. The space segment comprises a satellite or a constellation of satellites. A satellite is launched into orbit and then establishes communications with the ground segment. During regular operations, satellites may communicate through one or multiple ground stations [8].

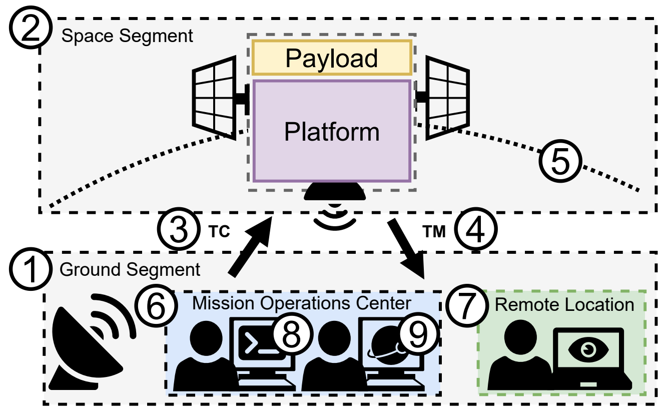


Fig. 1. The components commonly found in a satellite mission.

Telecommands (TC) ③ and Telemetry (TM) ④. The basic data flow between the space and ground segments are TC and TM [34]. TM is the data the satellite sends to the ground station which may contain the satellite's status or payload data [34]. TCs are used to operate the satellite and are transmitted and encapsulated in a space protocol (see Sec. II-E). 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 [8].

Orbital Pass ⑤. A satellite and its ground station can communicate *only* during an orbital pass. An orbital pass, or simply 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, e.g., mountains [35]. Passes can be predicted using two-line element (TLE) data [36].

Satellite Mission Operations ⑥. Satellite mission operations vary widely depending on the owning organization's budget, and technology [37]. Nevertheless, they share some commonalities, which we now describe.

A mission's operation involves a team of operators that use ground segment software (GSS) to operate a satellite(s) and ensure the mission's success [38]. Operations are carried out in a Mission Operations Center (MOC), where operators sit at their workstations to manage TCs sent to the satellite(s). Operators may also remotely operate satellites ⑦ by connecting to the ground segment using tools such as VNC (Virtual Network Computing) [7].

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 commands to be sent to the satellite to perform different functions, e.g., download payload data.

Ground (Segment) Software (GSS). GSS allows operators to carry out the mission's routine operations. GSS are very diverse. Some satellite missions develop their own GSS while others use open-source [39] or proprietary GSS [40]. 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 display TM. For example, ESA's SCOS-2000 is an MCS that provides generic functionality that can be customized for a specific mission operation [41]. Some missions develop their own MCS. For example, the SUCHAI mission developed their own MCS application [39].

Ground station control software (GSCS) ⑨ helps satellite operators track and visualize the satellite's orbit and provide information about each satellite's pass. For example, Gpredict is a popular open-source GSCS that performs real-time satellite tracking and orbit prediction [42].

D. Satellite Architecture

Satellite architectures are varied and complex [43]; 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 activities and the *payload*. The platform underpins the payload that fulfills the mission's purpose. Payloads differ depending on the satellite's mission and can range from measurement instruments to communications systems.

Platform: The platform is composed of custom-designed or off-the-shelf subsystems necessary for critical satellite operations. These include the Attitude Determination and Control System (ADCS) to maintain the satellite's orientation (i.e. attitude); 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.

These subsystems are controlled via TCs sent to the satellite. TCs may be interpreted by a central C&DH System or merely forwarded to the recipient subsystem [44]. This managerial duty is handled by the Flight Software (FS) running on an embedded system, for example, NASA's Core Flight System (cFS) [45]. Attackers aim to gain the ability to send TCs to the flight software as this may grant control over all subsystems.

Payload. The payload is the equipment that a satellite employs to fulfill its mission. Due to satellites' varied missions, payloads are heavily customized [46]. For example, if a satellite's mission is remote sensing, its payload may include an infrared camera [47].

E. Small Satellite Protocol Ecosystems

SmallSat missions can often be categorized by the adoption of protocols and their corresponding philosophies. Currently, there are two major protocol ecosystems which SmallSat missions can adopt: CSP and CCSDS.

Cubesat Space Protocol. The Cubesat Space Protocol (CSP) family of protocols is a one-stop solution for SmallSat missions [48]. CSP is implemented as an open-source C library called *libCSP* [48] and follows the TCP/IP model, including transport and routing protocols and multiple layer 2 interfaces such as I^{2}C (Inter-Integrated Circuit), and ZeroMQ (ZMQ) for transmission on TCP/IP networks [49].

In the context of a satellite mission, CSP connects the ground segment and satellite subsystems as part of a CSP network where each subsystem is identified as a node. Sending a TC is as simple as sending a CSP packet with the address of the corresponding subsystem node.

CCSDS Space Communication Protocols. The Consultative Committee for Space Data Systems (CCSDS) Space Communication is a set of standardized protocols used for different purposes in space communications [50]. A relevant CCSDS protocol for SmallSat TM and TC is the Space Packet protocol. This protocol is used in combination with the ECSS Packet Utilization Standard (PUS) [51] to define how TCs and TM are encoded and transported. PUS defines services (and thus sets of TC/TM) for functionality that satellite missions require, including large data transfer or event reporting [50].

F. Space Systems' Tactics, Techniques and Procedures (TTPs)

Tactics, Techniques, and Procedures (TTPs) describe a malicious actor's behavior in a structured scheme to understand how they might execute an attack [52], [53]. The SPACE-SHIELD matrix [54] is a framework used to standardize space systems' TTPs, for example, *ground segment compromise*.

III. Threat Model

Following Fig. 2, we assume that 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 the exposed network protocols discussed in Sec. II-E, which correspond to 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 as depicted in Fig. 4. 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. The threat model is further referenced in Sec. IV-E.

IV. HoneySat Framework Design

In this section, we explain the objectives we aim to achieve (Sec. IV-A), the design principles we follow to meet such objectives (Sec. IV-B), and the overall design of our Space (Sec. IV-D), and Ground (Sec. IV-C) segments.

A. HoneySat's Design Objectives

Our design aims to achieve the following objectives:

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

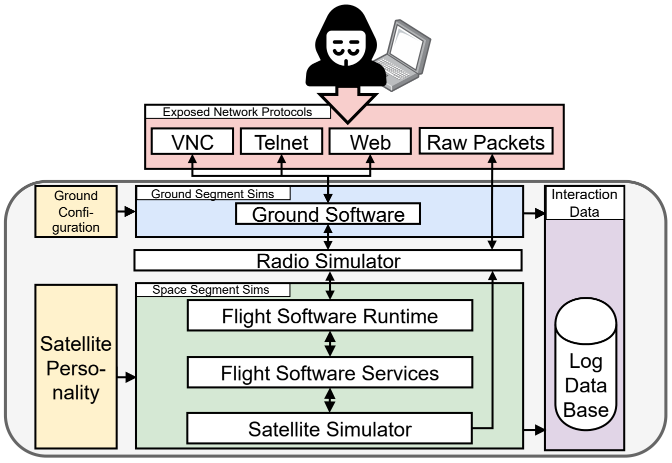


Fig. 2. HoneySat framework high-level architecture design.

DO-2 **Provide Deception.** As we discussed in Sec. I, honeypots' nature must remain *covert* to entice adversaries to interact 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. As such, HoneySat's third objective is extensibility and customizability.

B. HoneySat's Design Principles

To meet our objectives, we selected the following principles:

DP-1 **High-Interaction Simulation.** As we described in Sec. II-A, high-interaction honeypots give adversaries the same or almost identical interaction opportunities as a real satellite. 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 discussed in Sec. II-B, satellites track their orbit location, among others. These details must be simulated; 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. II, satellites are complex and diverse systems. To tackle this problem, we designed HoneySat to be modular. This design principle is based on our objective DO-3.

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 VII illustrates how HoneySat's simulation components match the components of a real satellite mission.

C. Ground Segment Design

Following Sec. II-C, the purpose of the HoneySat's ground segment (depicted in Fig. IV) 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 the design principle DP-3. We selected these protocols based on data obtained from our satellite operator survey, discussed in Sec. VI-C, which revealed that satellite operators do use remote access tools, such as web interfaces and screen sharing, to operate satellites.

2) Ground Software. As discussed in Sec. II-C, the ground software includes mission control software (MCS) and ground station control software (GSCS). We leverage existing ground software, e.g., Gpredict, allowing HoneySat to provide a high-interaction simulation following design principle DP-1.

3) Radio Simulator. As discussed in Sec. II-C, communication between the satellite and the ground station is possible only during a pass. The purpose of the Radio Simulator is to mimic real orbital passes by enabling and disabling communication between HoneySat's ground and space segment simulations at the appropriate times. The Radio Simulator design follows the design principles DP-2 and DP-3.

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

5) Logging Repository. This repository records 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 repository design follows design principle DP-3.

D. Space Segment Design

The purpose of the HoneySat's space segment is to mimic the spacecraft, as discussed in Sec. II-C. 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. II-D, 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.

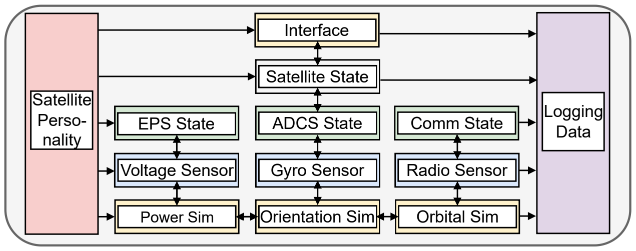


Fig. 3. High-level architecture of the Satellite Simulator. Due to space limitations, we do not show all the available sensors and simulations.

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 solves this problem by simulating all the necessary satellite subsystems, e.g., the Electrical Power System, and the sensors, e.g., a GPS receiver.

As shown in Fig. 3, the Satellite Simulator includes five module types: *simulation modules* (orange), *sensor modules* (blue), *subsystem state modules* (green), *satellite state modules* (gray) and *interface modules* (yellow).

Simulation Modules. These modules are abstractions of real-world processes required for a realistic satellite simulation (DP-2), e.g., the orbital simulation which uses orbital mechanics to calculate data like the satellite's orbital position.

These simulation modules can communicate with each other to facilitate proper functionality. For example, the power simulation queries the orbital simulation for data to determine if the satellite is positioned properly to draw power from its solar panels. The Satellite Simulator includes six simulations, the orbital, rotation, power system, thermal, magnetic, and payload simulations.

- *Orbital Simulation.* The orbital simulation calculates the satellite's orbit [36]. It uses the TLE data configured in the satellite personality and it generates multiple values such as, latitude, and longitude of the satellite. The payload, EPS, and magnetic simulations rely on data from this simulation.
- *Rotation Simulation.* This simulation calculates the satellite's attitude changes to provide its orientation by using 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 [55]. This simulation's data is used by payloads like the Red, Green, Blue (RGB) camera to point towards Earth's surface.
- *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.
- *Thermal Simulation.* This simulation calculates the satel-

lite's temperature based on the total thermal energy and a user-defined specific heat capacity and uses a radiation loss formula [56] 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 to calculate the Earth's magnetic field components.
- *RGB Camera Simulation.* This simulation replicates the functionality of an RGB camera pointed at Earth. It uses Earth observation satellite imagery from the U.S. Geological Survey (USGS) [57]. If a capture image command is issued, it will select and return an image. This simulation communicates with the orbital simulation to determine if satellite is in the presence of the sun.

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. II-D. A given subsystem state is made up of one or more sensors. For example, in Fig. 3 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. 3 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. 3, 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.

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 satellite 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 complying with our design principles DP-1 and DP-3.

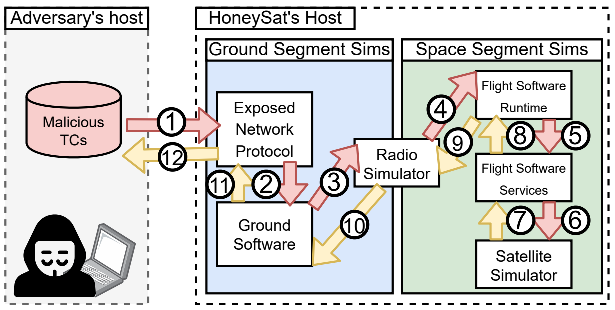


Fig. 4. HoneySat's Theory of Operation.

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 makes our framework easily customizable; thus following our design principle DP-3.

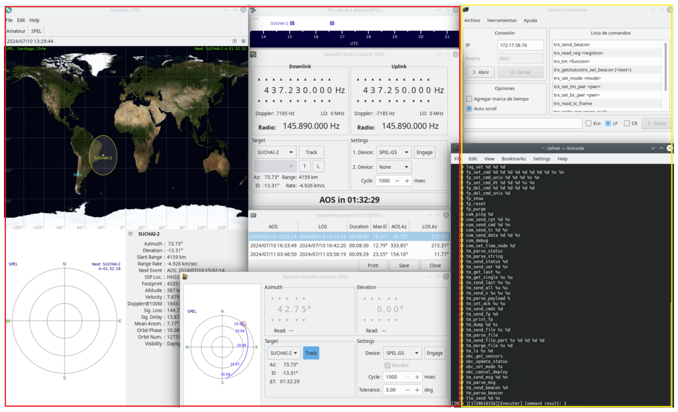
5) Logging Repository. We designed each space segment component to provide detailed logs. As shown in Fig. 3, all Satellite Simulator modules can send their own logs.

E. 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. 4.

An adversary gets initial access via one of the *Exposed Network Protocols* ①. On interacting 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 *Logging Repository*. 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* 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



```
trx_send_beacon
trx_read_reg <registro>
trx_tm <funcion>
trx_getstatustrx_set_beacon [<text>]
trx_set_mode <mode>
trx_set_tm_pwr <pwr>
trx_set_bc_pwr <pwr>
trx_read_tc_frame
```

```
log_set %d %d %d
fp_set_cmd %d %d %d %d %d %d %d %d %s %n
fp_set_cmd_unix %d %d %d %s %n
fp_set_cmd_dt %d %d %d %s %n
fp_del_cmd %d %d %d %d %d %d
fp_del_cmd_unix %d
fp_show
fp_reset
fp_purge
com_ping %d
com_send_rpt %d %s
com_send_cmd %d %n
com_send_tc %d %n
com_send_data %d %d %n
com_debug
com_set_time_node %d
tm_parse_status
tm_parse_string
tm_send_status %d
tm_send_var %d %s
tm_get_last %u
tm_get_single %u %u
tm_send_last %u %u
tm_send_all %u %u
tm_send_n %u %u %u
tm_parse_payload %
tm_set_ack %u %u
tm_send_cmds %d
tm_send_fp %d
tm_print_fp
tm_dump %d %s
tm_send_file %s %d
tm_parse_file
tm_send_file_part %s %d %d %d %d
tm_merge_file %s %d
tm_ls %s %d
obc_get_sensors
obc_update_status
obc_set_mode %s
obc_cancel_deploy
tm_send_msg %d %n
tm_parse_msg
tm_send_beacon %d
tm_parse_beacon
tle_send %d %s
[INFO][1720618158][Executer] Command result: 1
```

Fig. 5. The view an attacker would see upon connection to the VNC server. Ground Station Control Software (Red) Ecosystem/Mission Specific MCS (Yellow).

attacker or the ground software used by the attacker ⑦-⑫, and is also recorded in our *Logging Repository*.

V. HoneySat's Implementation

Having laid out HoneySat's design we now explain how we implemented the Satellite Simulator (Sec. V-A), the generic CSP mission honeypot (Sec. V-B), the ground segment (Sec. V-B1), and the space segment (Sec. V-B2).

A. 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.

B. Generic CSP Mission Honeypot

We implemented HoneySat to support the CSP ecosystem providing a generic CSP-based honeypot that can simulate any specific CSP-based mission.

1) Ground Segment Implementation: We now describe how we implemented the five ground segment simulations designed in Sec. IV-C. 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 a VNC server, a Telnet server, a Web server, and ZeroMQ-based access to the CSP Network. We implemented the VNC server using TigerVNC [58] and used PyZMQ [59] for the raw packet access. The Telnet server was implemented using Python to expose the command-line interface of the Ground Segment Software. The web interface allows for the customization of text presented to the attacker through configuration.

2) Ground Software. To implement the MCS 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. Additionally, we provide a desktop environment accessible via VNC that includes Gpredict and the SUCHAI MCS. Finally, we implemented a radio link and network delay simulation using a sleep function whenever the MCS receives data from the space segment simulations.

3) Radio Simulator. The Radio Simulator was implemented as a *ZMQ Hub* [60]. 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 the space segment. Depending on whether the satellite is currently passing, packets to the space segment are forwarded to the Flight Software Runtime. Additionally, the Radio Simulator dynamically simulates packet loss depending on the predicted elevation angle of the satellite (provided by the Satellite Simulator's orbital simulation). For example, if an attacker sends a telecommand, the Radio Simulator will use a probability value based on the elevation angle and decide to drop or forward the packet.

4) Ground Configuration. We implemented the ground configuration as settings that can be modified through a Docker compose file explained in Appendix A.

5) Logging Repository. This implementation uses the MongoDB database [61] 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. The logs are secured at points that are not observable by attackers, e.g., different Docker container, shown in Fig. 6, and the isolated MongoDB database is accessed by an append-only user.

2) Space Segment Implementation: We now describe how we implemented the four space segment simulations designed in Sec. IV-D. 1) the *Flight Software Runtime*, 2) the *Flight Software Services*, 3) the *Satellite Personality*,and 4) the *Logging Repository*.

1) Flight Software Runtime. We use the *SUCHAI Flight Software* (SUCHAI FS) [62], which has been deployed in four satellite missions [63] and provides flight software based on the CSP ecosystem.

2) Flight Software Services. The SUCHAI FS uses libCSP [48] and thus features the libCSP default services that present a generic attack surface. SUCHAI FS includes a set of telecommands which we modified to use the Satellite Simulator instead of querying real sensors.

3) Satellite Personality. The satellite personality was implemented as a Python class that holds multiple variables related to the space segment. It includes almost 50 configurable variables, such as the satellite's name.

4) Logging Repository. Like the ground segment's logging repository (Sec V-B1), the Satellite Simulator and the FS are connected to the MongoDB instance using a MongoDB client.

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

C. Summary of Existing and Newly Developed Software

HoneySat's implementation involved the integration of existing, modified, and newly developed software. *As-is software* includes TigerVNC, MongoDB, and Gpredict. *Modified software* includes the SUCHAI FS, SUCHAI MCS, the Telnet server, and the ZeroMQ Hub. *Newly developed software* includes the Satellite Simulator, including all 6 simulations, the web interface, the flight software services for FS integration, the satellite personality, the ground configuration, and all the necessary framework infrastructure, including Docker files, integration scripts, and configuration files.

VI. Evaluation

In order to evaluate HoneySat we start by listing three experimental questions designed to test its alignment with our design objectives. Next, we present four sets of experiments that provide empirical evidence confirming that our design objectives have been addressed as intended. We then describe each experiment's environment, methodologies, and results.

A. Experimental Questions

The following questions aim to determine if HoneySat meets the design objectives outlined in Sec. IV-A.

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. IV. This question is related to design objective DO-1 and is addressed in Sec. VI-B.

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

Q-3 **Can HoneySat be extended to support different SmallSat missions and hardware-in-the-loop simulations?**
HoneySat's customization is important because it would allow users of our framework to implement their own honeypots and even support satellite hardware integration. This question relates to design objective DO-3, and we answer it in Sec. VI-E and Sec. VI-F.

To answer the above research questions, we conducted five experiments. First, in Sec. VI-B we craft multiple attacks in a controlled environment to quantify the level of interaction that HoneySat provides. Second, in Sec. VI-C we conduct a survey with experienced satellite operators to evaluate HoneySat's realism. Third, in Sec. VI-D we deploy HoneySat and expose it to the Internet to test its deception capabilities. Fourth, in Sec. VI-E we test HoneySat's extensibility by integrating a completely different flight software. Fifth and final, in Sec. VI-F we test HoneySat's support for hardware-in-the-loop operations.

B. TTP Interaction Experiment

This experiment seeks to answer Q-1 by quantifying the interactions provided by HoneySat. To achieve this, we leveraged the SPACE-SHIELD matrix (Version 2.0) [54] which provides a collection of adversary tactics and techniques for space systems. 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 are applicable to HoneySat. From these tactics, HoneySat can feasibly offer up to 38 techniques as interactions for adversaries.

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 38 applicable 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. For example, the technique *System Service Discovery (ID T1007)* involves adversaries obtaining information about services using tools and OS utility commands. 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
```

Experiment Results. We successfully crafted and ran an interaction or exploit on HoneySat for 33 techniques out of the feasible 38 as depicted in Table II. The remaining 5 techniques were not implemented due to limitations in HoneySat's implementation. For example, the *Retrieve TT&C master/session keys (T2015.002)* technique depends on a cryptographic protocol implementation which HoneySat's flight software currently does not support. 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 VIII in Appendix B.

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 87% of the SPACE-SHIELD matrix techniques possible in a virtual satellite honeypot.
- HoneySat supports 100% of the SPACE-SHIELD matrix tactics.

C. SmallSat Operators Survey

Evaluating the realism of a satellite honeypot is challenging for two main reasons. First, as discussed in Sec. I, satellites, including SmallSats, are very diverse. Second, there is no

TABLE II
TACTICS AND TECHNIQUES SUPPORTED BY HONEYSAT.

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

established metric or tool, such as Nmap's OS detection [64], to quantify the level of realism of our honeypot.

To evaluate the realism and deception capabilities of HoneySat, we surveyed *experienced 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.

1) Survey Structure: We divided our survey into five sections. The first section collected the participants' background information, e.g., demographic data, the second section focused on their professional experience, the third section probed participants' satellite operation experience, e.g., how many missions they had operated.

The fourth survey section involved giving participants *hands-on access* to a *live instance of HoneySat*. In this section, participants were asked to perform 4 hands-on tasks that correspond to specific satellite mission operations using a live instance of HoneySat as depicted in Fig. 5. Each hands-on task was designed to feature different components of HoneySat by replicating a real-world satellite mission operation, as discussed in Section II. After participants completed each hands-on task, they answered questions designed to assess how realistic they found different aspects of the honeypot. Table III lists the satellite operations included in the survey.

Finally, in section five, once participants had interacted with multiple elements of our honeypot, the survey concluded with participants answering questions about HoneySat's overall realism and deception. Questions related to both the honeypot operation tasks and the overall evaluation used a 5-point Likert scale [65] to measure the participants' reactions.

2) Participants: We conducted the survey via Qualtrics and Zoom videoconferencing and distributed the survey directly to operators from previously identified missions. In total, we received responses from 10 satellite operators who have between 1 to 10 years of experience operating satellites and have operated between 1 to 5 unique missions. In terms of demographics, 20% (2/10) of the participants were female and 80% (8/10) male. 30% (3/10) belonged to the 18-24 age group, 40% (4/10) to the 25-34 age group, and 30% (3/10) to the 35-44 age group. In regards to geographic location, 40% (4/10) of participants were located in Europe, 20% (2/10) in North America and 40% (4/10) 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. Despite these challenges, our sample is diverse, including operators from Europe, North America, and South America, spanning industry, government, and academia, across 5 missions.

3) Methodology and Key Results: In the survey, we evaluated 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 SmallSat mission simulation.

Before describing the results, it is important to emphasize that the participants were *informed* that the system they interacted with was a simulation. We informed participants for two reasons. First, it was not feasible to synchronize participants' availability to take the survey with the timing of the real satellite's pass, which may happen only a few times per day and lasts only a few minutes. Second, claiming to provide access to a real satellite would itself be seen as unrealistic.

Ground Segment Realism. To understand if HoneySat's ground segment is realistic, participants interacted with HoneySat by performing different satellite operations (discussed in Sec. II-C), to showcase the ground segment components listed in Table III. One of these tasks is the *pass prediction* operation which involves calculating when and where a satellite will be within the communication range of a specific ground station. This operation is performed using the ground station control software. After the participants performed this operation on HoneySat, we asked them to rate their perceived level of realism. 90% of the participants strongly agreed and 10% agreed that the *pass prediction* operation they performed on HoneySat resembles that of a real mission.

Another relevant operation is the *telecommand scheduling* operation which involves the planning and queuing of the commands to be sent to the satellite during a pass. Again, after performing this operation on HoneySat, 90% of the participants strongly agreed and 10% agreed that the telecommand scheduling operation performed resembles that of a real

TABLE III
SUMMARY OF QUESTIONS IN SECTION VI-C, SATELLITE HONEYPOT OPERATION TASKS OF THE SURVEY.

Satellite Operation	Evaluated Component	No. Qts.
Telecommand Scheduling	Mission Control SW	5
Pass Prediction	Ground Station Control SW	5
Telemetry Download	Satellite Simulator	7
Ping Test	Radio Simulator	5

mission mentioning the use of a command line interface-based mission control software as a contributing factor.

In summary, according to these results, the vast majority of participants, who are *experienced satellite operators*, perceive the ground segment simulated by HoneySat as highly resembling a real satellite mission.

Space Segment Realism. Similarly, to understand HoneySat's space segment realism level, we gave participants hands-on access to a live instance of HoneySat and asked them to perform different operations that make use of HoneySat's space segment simulations (discussed in Sec. IV-D).

For example in the *telemetry download* operation, participants issued multiple telecommands to HoneySat's space segment simulation to download a plethora of telemetry data, including EPS, temperature and ADCS data, which was generated in real time by our Satellite Simulator.

After the participants performed the telemetry download operation, we asked them to rate their perceived level of realism of the telemetry output shown during the operation. 90% of the participants agreed or strongly agreed that the telemetry shown during the live *telemetry download* operation resembles that of a real mission, with the temperature telemetry achieving a 100% strongly agree response rate with some participants mentioning that the temperature values resemble a normal operation and align with the expected temperature values of a satellite. These results indicate that the telemetry generated by the Satellite Simulator is considered highly realistic by the vast majority of participants.

In summary, according to these results, the vast majority of participants, who are *experienced satellite operators*, perceive the space segment simulated by HoneySat as highly resembling a real satellite mission.

HoneySat's Overall Realism. Finally, to understand HoneySat's overall realism, we asked the participants a series of overall evaluation questions after they had interacted with HoneySat. For example, we asked participants to evaluate the realism of the telecommands used in all hands-on tasks. 70% strongly agreed, 20% agreed and 10% neither agreed nor disagreed that HoneySat's telecommands are realistic. Additionally, when asked if they would be able to distinguish between HoneySat's satellite mission simulation and a real mission, 70% strongly agreed, 20% agreed and 10% neither agreed nor disagreed that they would *not* be able to and again to emphasize that these are *experienced satellite operators*.

Overall, our survey results provide evidence for answering question Q-2 in the affirmative and design objective DO-2 as achieved. These findings serve as empirical evidence of HoneySat's fidelity when simulating a real satellite mission. Our survey's questions and results are available online[3].

[3]https://github.com/HoneySat/honeysat-survey-data.

TABLE IV
HONEYSAT INTERNET DEPLOYMENTS' DETAILS.

Deployment Type	Satellite Personality	HoneySat IP Location	Duration (months)
Cloud	PIXL-1	Germany	6
Cloud	PIXL-1	Germany	6
Cloud	ACS3	USA	6
Cloud	ACS3	USA	6
On-prem	SUCHAI-2	University of Chile	12

Key finding Q-2

90% of surveyed satellite operators agreed with the statement: *"I would not be able to distinguish between HoneySat's satellite honeypot system from the real CubeSat satellite mission it is based on."*

D. Internet Interaction Experiment

This experiment explores HoneySat's capabilities to entice external actors by deploying it over the Internet.

Experiment Methodology. We leveraged HoneySat's customization and deployability features to deploy five instances of HoneySat over the Internet and exposed TCP ports 23 for Telnet, 80 for the web interface, and 5901 for VNC.

Deployment. Initially, we deployed our first HoneySat instance to simulate the SUCHAI-2 SmallSat on the premises of the University of Chile, where the mission's real ground segment is. This on-premises deployment provides the most convincing environment to adversaries. After coordinating with the University of Chile's local stakeholders for three months, we deployed HoneySat on an onsite server using a University of Chile IP address starting on July 2024.

Although we went to great lengths to make our deployment as realistic as possible, after a few months, our honeypot did not entice any external actors. Therefore, in January 2025, we deployed four additional HoneySat instances on cloud servers. We configured two instances to simulate NASA's Advanced Composite Solar Sail System (ACS3) CubeSat [66], and two simulating DLR's PIXL-1 CubeSat [67]. We selected these two particular CubeSats because they use the CSP ecosystem (Sec. II-E) and are currently in-orbit and operational. We ensured these additional deployments were believable by configuring their satellite personalities and giving them public IP addresses in the regions belonging to the satellite mission origin, namely Germany and the United States. Table IV summarizes all five HoneySat deployment details.

While IP assignment can affect our honeypots' covert status [18], the increasing use of cloud and web-based ground segment services, such as AWS Ground Station [68] and YAMCS [69], has made cloud-hosted satellite operations increasingly commonplace. As a result, a cloud deployment no longer inherently signals inauthenticity to adversaries. Furthermore, the part of the Ground Segment that we expose on cloud IPs is not dependent on being collocated with the GS and thus may as well be hosted with a cloud provider.

TABLE V
EXPOSED TELNET INTERACTIONS RECEIVED.

Date	Satellite Personality	HoneySat IP	Attacker IP	Cmds Received	Time
Jan 18, 2025	ACS3	USA	Egypt	4	2 hr
Jan 24, 2025	PIXL-1	Germany	Sweden (Tor)	6	4 min
Jan 24, 2025	ACS3	USA	France (Tor)	9	5 min
Apr 3, 2025	ACS3	USA	USA	8	3 min

Experiment Results. We collected several gigabytes of network traffic data on ports 23, 80 and 5901 for each honeypot, however most of these data was generated by crawlers and bots and not satellite-specific. But, on January 18th, 2025, we received our first space-specific interaction via Telnet and in the next few months we received more. Overall, three of our honeypots successfully enticed external actors and captured four distinct interaction sessions (shown in Table V) all via Telnet. The four interaction sessions yielded 22 flight software-specific commands. These flight software-specific commands show that the adversaries that interacted with HoneySat were purposefully trying to exploit our honeypot using commands that the flight software recognized.

Across the four Internet interaction sessions we observed, the first lasted two hours and originated from a non-Tor IP in Egypt. The next two sessions occurred within a week, shared their command set (e.g., `help`, `ls`, and `fp_show`), and transitioned to Tor exit nodes in Sweden and France. These three factors suggest that the same adversary returned using anonymized infrastructure after initial reconnaissance. The first session may have ended after the actor gathered sufficient information. In contrast, the fourth and final interaction in April, separated by over two months, from a non-Tor U.S. address, and using a very different command set appears unrelated and likely originates from a different actor. Overall, the first three interactions are best interpreted as a single adversary adapting its anonymity strategy, whereas the last session was likely initiated by a second actor.

We now describe one of the most complex interaction sessions HoneySat captured and analyze it in terms of the SPACE-SHIELD matrix (discussed in Sec. II-F). In this session, the adversary targeted one of the ACS3 honeypots located in the United States. First, they connected to the exposed Telnet server thus getting access to the ground segment, this corresponds to the *Ground Segment Compromise (ID T1584.001)* technique. Then they issued the *"help"* and *"test"* commands which list the available commands supported by the MCS and flight software; these commands correspond to the *Gather Victim Mission Information (ID T2002)* technique. Later, the adversary issued the *"tm dump 0x0000"*, which writes telemetry data to a file, showing an attempt to extract mission data, which matches the *Exfiltration Over TM Channel (ID T2022)* technique. Next, they sent the *"com_ping"* command which attempts to initiate an interaction with the target spacecraft, matching the *Active Scanning (RF/Optical) (ID T2001)* technique. The adversary again implemented the *Exfiltration Over TM Channel (ID T2022)* technique by issuing the *"tm_parse_beacon"* command which prints beacon telemetry (lightweight periodic telemetry) and the *"obc_get_sensor"* command which instructs the OBC to print a particular sensor's data. Finally, the adversary attempted to tamper with the spacecraft's OBC by issuing the *"obc_update_status"* command. This command updates the OBC status variables to any arbitrary value, thus matching the *Modification of On Board Control Procedures (ID T2010)* technique.

Overall, this interaction session alone involved a chain of five SPACE-SHIELD matrix techniques. However, after analyzing all four interaction sessions, we identified the *Spacecraft's Components Discovery (ID T2034)* technique, totaling six SPACE-SHIELD techniques based on real-world data captured by HoneySat after successfully enticing and deceiving human adversaries.

While our dataset is relatively small, its significance becomes evident when contextualized. For example, according to a 2018 Open Platform Communications (OPC) group report [70], there are an estimated 47 million OPC-enabled ICS devices deployed worldwide, compared to 11,000 operational satellites in 2025 [71]. This represents about *4,700 ICS devices for every one satellite*. In other words, attackers have orders of magnitude more opportunities to target ICS devices than satellites. Thus, making our dataset a rare and valuable contribution to understanding space systems' TTPs.

In addition to its rarity, our dataset is the *first* empirical real-world dataset of TTPs against satellites. As such, it not only provides the first view into how adversaries interact with satellite systems, but also establishes a reproducible baseline reference for future studies. Lastly, our contribution complements prior work that provides limited insights, e.g., no specific telecommands [72].

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

Key findings Q-2

- HoneySat deceived human adversaries who sent *22 satellite flight software-specific commands*.
- The interactions collected by our HoneySat deployments comprise six SPACE-SHIELD techniques.

E. Case Study: Extending HoneySat to CCSDS Ecosystem

In Sec. V-B 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. II-E) by adding a *second ecosystem* to HoneySat, namely the CCSDS ecosystem.

Experiment Description. We selected CCSDS because it is a standard protocol suite used by other SmallSats [8]. To accomplish this, we leveraged an open-source CCSDS ecosystem-based flight software framework, RAC-

TABLE VI
SMALLSAT ECOSYSTEMS (2) AND EXAMPLE SMALLSATS (3) INTEGRATED TO HONEYSAT.

Honeypot Framework	Generic SmallSat	Ground Configuration	Institution
HoneySat	CSP	ACS3	NASA
		PIXL-1	DLR
		SUCHAI-2	U of Chile
	CCSDS	PUS	N/A

COON OS [73], and YAMCS, an open-source Mission Control software framework with built-in support for PUS [69].

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

Regarding the ground segment, we implemented the *exposed network protocol* using YAMCS' built-in web interface. For the *mission control software*, we used YAMCS's built-in Mission Control Software [69]. For the *radio simulator*, we used RACCOON's communication application. For both the *ground configuration* and the *logging repository*, we again used YAMCS built-in features.

For the space segment, we implemented the *flight software runtime* using the RACCOON framework. 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.

Experiment Results. We successfully extended HoneySat to support the CCSDS ecosystem. The implementation was completed by a graduate student with no prior satellite-related experience in the span of two weeks. However, the majority of the effort involved in extending HoneySat to support the CCSDS ecosystem was dedicated to understanding the ecosystem itself, and analyzing 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 modules such as the Satellite Simulator.

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

Key findings Q-3

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

F. Case Study: Hardware-in-the-loop Experiment

This experiment is designed to produce evidence of the robustness of HoneySat to support real-world satellite hardware-in-the-loop (HIL) operations.

Experiment Description. In this experiment we integrated HoneySat with an *in-orbit, operational SmallSat* mission. To achieve this, we collaborated with an aerospace company that develops hardware subsystems for SmallSats. The experiment involved sending a telecommand from HoneySat's ground segment simulation which was then sent to the in-orbit SmallSat, which would then process the telecommand. Due to security restrictions, we are unable to disclose the name of the in-orbit SmallSat which we refer to as SmallSat X.

Experiment Methodology. In order to achieve the HIL integration, we customized both HoneySat and SmallSat X's mission. For HoneySat, we modified the Satellite Simulator to support SmallSat X's mission control software (YAMCS). Specifically, we deployed a proxy server that connected the Satellite Simulator to the SmallSat X's mission production YAMCS instance. Conversely, the SmallSat X's environment was customized by aerospace company's team by deploying a script in their YAMCS instance to complete the integration. Once the integration was completed, we coordinated with the aerospace company to send a telecommand from HoneySat during one of SmallSat X's passes.

Experiment Results. The telecommand successfully reached SmallSat X during one of the passes and HoneySat received the appropriate telemetry. Due to security restrictions, we are not able to disclose details on the telecommand and the telemetry received. However, the entire communication was initiated by HoneySat which in turn received sanitized telemetry from SmallSat X, effectively closing the loop in the HIL experiment. Additionally, the aerospace company provided us with a snippet of radio signal data shown in Fig. 7, which confirms that the experiment was successful. In summary, these results highlight HoneySat's robust extensibility features and provide strong evidence for answering question Q-3 in the affirmative and design objective DO-3 as achieved. These findings serve as empirical evidence of HoneySat's fidelity when interacting with a real satellite mission.

Key findings Q-3

HoneySat was successfully integrated into a satellite hardware-in-the-loop operation and communicated with an *in-orbit, operational SmallSat*.

VII. DISCUSSION AND FUTURE WORK

Challenges of Creating the First Satellite Honeypot. During HoneySat's design and implementation we encountered and overcame two main challenges which stemmed from fundamental differences between satellites and other Cyber-Physical Systems (CPS), e.g., ICS.

First, *time and link-constrained communication.* Unlike ICS honeypots, which assume continuous network connectivity and stable control loops, e.g., scan cycles, satellite communication occurs only during orbital passes with variable duration and packet loss. To overcome this challenge, we developed the Radio Simulator discussed in Sec. IV-C.

Second, *dynamic physics-aware simulation.* ICS honeypots simulate sensor readings through simple physics loops (e.g., tank level, valve position). In contrast, a satellite honeypot must dynamically maintain physically coherent telemetry across interdependent subsystems such as attitude, and power. We addressed this by creating the *Satellite Simulator*, which simulates subsystems and sensors that communicate between each other and generate data in real time.

Satellite Honeynets. To increase attacker engagement and reflect emerging satellite networks [74], multiple HoneySat instances can be deployed as a honeynet [75]. Each instance could simulate different satellites and communicate through virtual inter-satellite links (ISLs). In addition, HoneySat could also be part of a honeynet connected to other ground mission infrastructure such as database servers and radio equipment. Due to HoneySat's extensibility, these honeynet scenarios could be implemented in the future.

Satellite Honeypots' Fingerprinting. Based on our experience designing, implementing and evaluating HoneySat we now discuss some qualitative detection vectors that can guide future satellite honeypot fingerprinting research.

Space Protocol Uniformity. The use of default space protocol configurations may be used for fingerprinting. For example, CSP deployments built directly from libCSP expose identical port assignments and diagnostic services, e.g. ping.

Perfectly periodic pass timing. Real satellites rarely follow identical pass schedules; orbital perturbations, TLE drift, and operational delays introduce small timing variations. A honeypot that enables communication at perfectly fixed intervals or with constant latency can thus be fingerprinted. To avoid this, HoneySat periodically downloads the latest TLE data to keep the orbital simulation from drifting. This, on the other hand, causes a jump in the orbit once the new TLE data is used to base the simulation on.

Telecommand implementation. A given FS may have completely different telecommand names and functions depending on the satellite and thus a mismatch may reveal the honeypot. Additionally, telecommands may have revealing features that print information about the underlying honeypot host, e.g., Ubuntu, which may reveal the simulation.

A. Initial Satellite Honeypot Anti-Detection Techniques

Building upon lessons learned from HoneySat's development and from other CPS honeypots [76], [77], we identify concrete strategies to enhance HoneySat's and future satellite honeypots' resistance to fingerprinting and detection.

a) Improve Environmental Diversity: Previous research shows that static structural layouts and repeated host fingerprints enable easy detection [77]. Future satellite honeypots and honeynets can incorporate diversity in its ground segment topology, and network services. Following the Purdue-style segmentation [78] used in ICS honeypots, different HoneySat modules (mission control, station control, payload) can be isolated behind realistic network layers such as routers and firewalls.

b) Implement Adaptive Reconfiguration: Honeypot adaptive reconfiguration occurs when a honeypot recognizes that it is being probed or scanned, e.g., Nmap probes, and reconfigures itself to avoid detection [79], [76]. Adapting this technique to satellite honeypots would involve both the ground and space segments. For example, the honeypot could modify exposed services, e.g., CSP over ZeroMQ to appear as a different ground station node. Additionally, different telemetry datasets could be rotated, or switched between low interaction and high interaction versions of the honeypot [80].

c) Use Real hardware and software identifiers: Future satellite honeypots could use real hardware identity and fingerprints such as MAC Organizationally Unique Identifiers (OUIs), and telemetry headers such as APIDs (Application Process Identifiers) [81]. Obtaining these details is non-trivial as they are often proprietary or absent from public documentation, and may require reverse-engineering, or careful analysis of captured mission traffic.

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 [57]). 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. Originally designed as a honeypot, our framework can potentially 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 cybersecurity training exercises.

VIII. Conclusion

Although we have yet to witness a Stuxnet-like cyberattack on space systems, security researchers need to develop effective countermeasures to secure satellites. In this paper, we introduced HoneySat, the first satellite honeypot that simulates a satellite mission and provided evidence that our honeypot can obtain real-world interaction data and can be extended with real-world satellite software and hardware. Finally, we hope that security researchers use HoneySat's open-source implementation as a foundation not only for satellite honeypot deployments but also for simulation, and training applications.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their helpful suggestions towards improving this paper. The project underlying this paper was funded by the Federal Ministry of Transport (BMV) under the code 45AVF5A011 and was partially supported by NSF awards No. 2131263 and 2232911, and by the US Department of Transportation (US-DOT) CYBER-CARE Grant No. 69A3552348332. We further thank the Saarbrücken Graduate School of Computer Science for their funding and support. The authors are responsible for the content of this publication.

IX. Ethical Considerations

In this paper, we consider the ethical consequences and possible negative outcomes of the satellite operator user study and the HoneySat deployments. We now discuss the stakeholders, potential risks, and how we mitigated those risks.

A. Satellite Operators User Study

Prior to conducting any research, the survey protocol, and materials were submitted for review by our Institutional Review Board (IRB). The protocol received "exempt status", indicating no more than minimal risk.

Stakeholder Identification. The primary stakeholders in the survey are *survey participants*.

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) was collected, and the responses remain 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, 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."

B. 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.

Risk Mitigation. The main risk for the infrastructure owners is that external actors will use our honeypot as a stepping stone to breach their infrastructure. For the on-prem deployments, we worked with the local IT administrators to ensure that we took all the necessary precautions to avoid this scenario. First, we were provided 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.

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 [82]. 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.

References

[1] United States Space Force, "Global positioning system ¿ space operations command (spoc) ¿ display," Feb. 2023. [Online]. Available: https://www.spoc.spaceforce.mil/About-Us/Fact-Sheets/Display/Article/2381726/global-positioning-system

[2] Gunter's Space Page, "Uwe-1 (universität würzburg's experimentalsatellit-1)," https://space.skyrocket.de/doc_sdat/uwe-1.htm, 2005, accessed: 2025-08-06.

[3] M. Holmes, "The growing risk of a major satellite cyber attack," May 2023. [Online]. Available: https://interactive.satellitetoday.com/the-growing-risk-of-a-major-satellite-cyber-attack/

[4] V. Charlotte and P. Walter, "A world without satellite data as a result of a global cyber-attack," *Space Policy*, vol. 59, p. 101458, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0265964621000503

[5] J. Drmola and T. Hubik, "Kessler syndrome: System dynamics model," *Space Policy*, vol. 44, pp. 29–39, 2018.

[6] M. Semanik and P. Crotty, "U.s. private space launch industry is out of this world," Nov. 2023. [Online]. Available: https://www.usitc.gov/publications/332/executive_briefings/ebot_us_private_space_launch_industry_is_out_of_this_world.pdf

[7] S. L. O. California Polytechnic State University, "Earth station - polysat," Apr. 2024. [Online]. Available: https://www.polysat.org/earth-station

[8] J. Willbold, M. Schloegel, M. Vögele, M. Gerhardt, T. Holz, and A. Abbasi, "Space odyssey: An experimental software security analysis of satellites," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1–19.

[9] T. Scharnowski, F. Buchmann, S. Wörner, and T. Holz, "A case study on fuzzing satellite firmware," in *Workshop on the Security of Space and Satellite Systems (SpaceSec)*, 2023.

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

[11] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the sixth annual IEEE SMC information assurance workshop*. West Point, NY, USA: IEEE, 2005, pp. 29–36.

[12] F. Cohen, "The use of deception techniques: Honeypots and decoys," *Handbook of Information Security*, vol. 3, no. 1, pp. 646–655, 2006.

[13] ——, "Deception toolkit," Mar. 1998. [Online]. Available: http://all.net/dtk/

[14] L. Franceschi-Bicchierai, "Thousands of new honeypots deployed across israel to catch hackers," Nov. 2023. [Online]. Available: https://techcrunch.com/2023/11/20/thousands-of-new-honeypots-deployed-across-israel-to-catch-hackers/

[15] M. Burgess, "A clever honeypot tricked hackers into revealing their secrets," Aug. 2023. [Online]. Available: https://www.wired.com/story/hacker-honeypot-go-secure/

[16] S. Hilt, F. Maggi, C. Perine, L. Remorin, M. Rösler, and R. Vosseler, "Caught in the act: Running a realistic factory honeypot to capture real threats," *Trend Micro Research*, 2020.

[17] Shared Threat Intelligence for Network Gatekeeping and Automated Response (STINGAR), "About - stingar," Apr. 2024. [Online]. Available: https://stingar.security.duke.edu/about-2/

[18] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, "Honeyplc: A next-generation honeypot for industrial control systems," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 279–291. [Online]. Available: https://doi.org/10.1145/3372297.3423356

[19] B. Acharya, M. Saad, A. E. Cinà, L. Schönherr, H. D. 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: IEEE Computer Society, may 2024. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00156

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

[21] R. Bisping, J. Willbold, M. Strohmeier, and V. Lenders, "Wireless signal injection attacks on VSAT satellite modems," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 6075–6091. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/bisping

[22] G. Kavallieratos and S. Katsikas, "An exploratory analysis of the last frontier: A systematic literature review of cybersecurity in space," *International Journal of Critical Infrastructure Protection*, p. 100640, 2023.

[23] J. Willbold, M. Schloegel, R. Bisping, M. Strohmeier, T. Holz, and V. 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, 2024, pp. 288–299.

[24] NASA, "What are smallsats and cubesats?" [Online]. Available: https://www.nasa.gov/what-are-smallsats-and-cubesats/

[25] J. Vestergaard, "mushorg/conpot: Ics/scada honeypot," Mar. 2024. [Online]. Available: https://github.com/mushorg/conpot

[26] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Boston, MA, USA: Pearson Education, 2007.

[27] G. Marra, U. Planta, P. Wüstenberg, and A. Abbasi, "On the feasibility of cubesats application sandboxing for space missions," in *Workshop on the Security of Space and Satellite Systems (SpaceSec)*, 2024.

[28] M. Oosterhof, "cowrie/cowrie: Cowrie ssh/telnet honeypot https://cowrie.readthedocs.io," Apr. 2024. [Online]. Available: https://github.com/cowrie/cowrie

[29] N. Ilg, P. Duplys, D. Sisejkovic, and M. Menth, "A survey of contemporary open-source honeypots, frameworks, and tools," *Journal of Network and Computer Applications*, vol. 220, p. 103737, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S108480452300156X

[30] J. Nazario, "paralax/awesome-honeypots: an awesome list of honeypot resources," Mar. 2024. [Online]. Available: https://github.com/paralax/awesome-honeypots

[31] M. Lucchese, F. Lupia, M. Merro, F. Paci, N. Zannone, and A. Furfaro, "Honeyics: A high-interaction physics-aware honeynet for industrial control systems," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ser. ARES '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3600160.3604984

[32] J. Daubert, D. Boopalan, M. Mühlhäuser, and E. Vasilomanolakis, "Honeydrone: A medium-interaction unmanned aerial vehicle honeypot," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–6.

[33] M. Conti, F. Trolese, and F. Turrin, "Icspot: A high-interaction honeypot for industrial control systems," in *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, 2022, pp. 1–4.

[34] The European Space Agency, "Esa - telemetry & telecommand," Mar. 2013. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Telemetry_Telecommand

[35] L. Wood, "Introduction to satellite constellations: orbital types, uses and related facts," Jul. 2006. [Online]. Available: https://savi.sourceforge.io/about/lloyd-wood-isu-summer-06-constellations-talk.pdf

[36] D. A. Vallado and P. J. Cefola, "Two-line element sets–practice and use," in *63rd International Astronautical Congress*. Naples, Italy: International Astronautical Federation, 2012, pp. 1–14.

[37] A. Salces and J. Javier, "Maya-1: Cube satellite latest pinoy venture into space," *Philippine Daily Inquirer*, Jul 2018, accessed: 2025-08-06.

[38] NASA Goddard Space Flight Center, "Flight training: Introduction." [Online]. Available: https://solc.gsfc.nasa.gov/modules/missionops/mainMenu_textOnly.php

[39] Space and Planetary Exploration Laboratory, University of Chile, "SPEL / SUCHAI-Flight-Software · GitLab," Feb. 2024. [Online]. Available: https://gitlab.com/spel-uchile/suchai-flight-software

[40] GomSpace, "Nanocom ms100 datasheet," Mar. 2021. [Online]. Available: https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanocom-ms100-13.pdf

[41] M. Merri, A. Ercolani, D. Guerrucci, V. Reggestad, and D. Verrier, "Cutting the cost of esa mission ground software," May 2007. [Online]. Available: https://www.esa.int/esapub/bulletin/bulletin130/bul130g_merri.pdf

[42] A. Csete, "Gpredict: Free, real-time satellite tracking and orbit prediction software," Dec. 2023. [Online]. Available: https://oz9aec.dk/gpredict/

[43] W. D. Ivancic, "Architecture and system engineering development study of space-based satellite networks for nasa missions," in *2003 Aerospace Conference*, no. NASA/TM-2003-212187, 2003.

[44] B. D. Yost, "Nasa ssri knowledge base — detailed design and analysis ¿ subsystem design ¿ command and data handling," Jun. 2023. [Online]. Available: https://s3vi.ndc.nasa.gov/ssri-kb/topics/32/

[45] D. McComas, J. Wilmot, and A. Cudmore, "The core flight system (cfs) community: Providing low cost solutions for small spacecraft," in *Annual AIAA/USU Conference on Small Satellites*. Logan, UT: Utah State University, University Libraries, 2016.

[46] The European Space Agency, "Esa - about payload systems," Apr. 2024. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/About_Payload_Systems

[47] NASA, "What is remote sensing? — earthdata," Aug. 2019. [Online]. Available: https://www.earthdata.nasa.gov/learn/backgrounders/remote-sensing

[48] Y. Shoji, "libcsp/libcsp," Feb. 2024. [Online]. Available: https://github.com/libcsp/libcsp

[49] The ZeroMQ authors, "ZeroMQ — get started," 2024. [Online]. Available: https://zeromq.org/get-started/

[50] M. M. 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].

[51] J.-F. Kaufeler, "The esa standard for telemetry and telecommand packet utilisation: Pus," in *NASA. Goddard Space Flight Center, Third International Symposium on Space Mission Operations and Ground Data Systems, Part 2*, 1994.

[52] M. Raza, "What are ttps? tactics, techniques & procedures explained," Apr. 2024. [Online]. Available: https://www.splunk.com/en_us/blog/learn/ttp-tactics-techniques-procedures.html

[53] National Institute of Standards and Technology (NIST), "tactics, techniques, and procedures (ttp) - glossary," Apr. 2024. [Online]. Available: https://csrc.nist.gov/glossary/term/tactics_techniques_and_procedures

[54] E. S. Agency, "ESA SPACE-SHIELD," 2023. [Online]. Available: https://spaceshield.esa.int/#

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

[56] S. B. Giddings, "Hawking radiation, the stefan–boltzmann law, and unitarization," *Physics Letters B*, vol. 754, pp. 39–42, 2016.

[57] U.S. Geological Survey, "Earthexplorer — u.s. geological survey," Nov. 2022. [Online]. Available: https://www.usgs.gov/tools/earthexplorer

[58] P. Ossman, "Tigervnc/tigervnc: High performance, multi-platform vnc client and server," Jul. 2024. [Online]. Available: https://github.com/TigerVNC/tigervnc

[59] B. E. Granger and M. Ragan-Kelley, "Pyzmq documentation," Apr. 2024. [Online]. Available: https://pyzmq.readthedocs.io/en/latest/

[60] Y. Shoji, "The Protocol Stack — Cubesat Space Protocol," 2024. [Online]. Available: https://libcsp.github.io/libcsp/protocolstack.html

[61] MongoDB, Inc., "Mongodb: The developer data platform — mongodb," Apr. 2024. [Online]. Available: https://www.mongodb.com/

[62] C. E. Gonzalez, C. J. Rojas, A. Bergel, and M. A. Diaz, "An architecture-tracking approach to evaluate a modular and extensible flight software for cubesat nanosatellites," *IEEE Access*, vol. 7, pp. 126 409–126 429, 2019.

[63] C. Garrido, E. Obreque, M. Vidal-Valladares, S. Gutierrez, and M. 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. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2023/all2023/56/

[64] Nmap, "Os detection," Jan. 2025. [Online]. Available: https://nmap.org/book/man-os-detection.html

[65] Qualtrics, "What is a likert scale?" Jan. 2025. [Online]. Available: https://www.qualtrics.com/experience-management/research/likert-scale/

[66] NASA, "Advanced composite solar sail system (acs3)," Jan. 2025. [Online]. Available: https://www.nasa.gov/mission/acs3/

[67] eoPortal, "Pixl-1 / formerly cubel or osiris4cubesat," Jan. 2025. [Online]. Available: https://www.eoportal.org/satellite-missions/pixl-1

[68] Amazon Web Services, "Satellite as a service - aws ground station - aws," Mar. 2024. [Online]. Available: https://aws.amazon.com/ground-station/

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

[70] ARC Advisory Group, "Opc installed base insights," OPC Foundation, Tech. Rep., 2018, accessed October 2025. [Online]. Available: https://opcfoundation.org/wp-content/uploads/2018/02/ARC-Report-OPC-Installed-Base-Insights.pdf

[71] M. Wall. (2024, July) How many satellites could fit in earth orbit—and how many do we really need? Accessed October 2025, Live Science. [Online]. Available: https://www.livescience.com/space/space-exploration/how-many-satellites-could-fit-in-earth-orbit-and-how-many-do-we

[72] U.S.-China Economic and Security Review Commission, "2011 annual report to congress," U.S. Government Printing Office, Tech. Rep., 2011, accessed October 2025. [Online]. Available: https://www.uscc.gov/sites/default/files/annual_reports/annual_report_full_11.pdf

[73] P. W. José Manual Diez, Fabian Krech, "Raccoon os," https://gitlab.com/rccn, [Accessed 20-01-2025].

[74] S. Ma, Y. C. Chou, H. Zhao, L. Chen, X. Ma, and J. Liu, "Network characteristics of leo satellite constellations: A starlink-based measurement from end users," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[75] L. Salazar, E. López-Morales, J. Lozano, C. Rubio-Medrano, and A. A. Cárdenas, "Icsnet: A hybrid-interaction honeynet for industrial control systems," in *Proceedings of the Sixth Workshop on CPS&IoT Security and Privacy*, ser. CPSIoTSec'24. New York, NY, USA: Association for Computing Machinery, 2024, p. 68–79. [Online]. Available: https://doi.org/10.1145/3690134.3694813

[76] J. Uitto, S. Rauti, S. Laurén, and V. Leppänen, "A survey on anti-honeypot and anti-introspection methods," in *World Conference on Information Systems and Technologies*. Springer, 2017, pp. 125–134.

[77] V. Tay, X. Li, D. Mashima, B. Ng, P. Cao, Z. Kalbarczyk, and R. K. Iyer, "Taxonomy of fingerprinting techniques for evaluation of smart grid honeypot realism," in *2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2023, pp. 1–7.

[78] E. López-Morales, U. Planta, C. Rubio-Medrano, A. Abbasi, and A. A. Cardenas, "SoK: Security of programmable logic controllers," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 7103–7122. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/lopez-morales

[79] R. Gabrys, D. Silva, and M. Bilinski, "Honeygan pots: A deep learning approach for generating honeypots," 2024. [Online]. Available: https://arxiv.org/abs/2407.07292

[80] S. Kyung, W. Han, N. Tiwari, V. H. Dixit, L. Srinivas, Z. Zhao, A. Doupé, and G.-J. Ahn, "Honeyproxy: Design and implementation of next-generation honeynet via sdn," in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.

[81] T. C. C. for Space Data Systems (CCSDS), "Space Packet Protocol," 2020. [Online]. Available: https://public.ccsds.org/Pubs/133x0b2e1.pdf

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

[83] Docker Inc., "Docker: Accelerated container application development," Apr. 2024. [Online]. Available: https://www.docker.com/

Appendix A
Deployment and Security Hardening Implementation

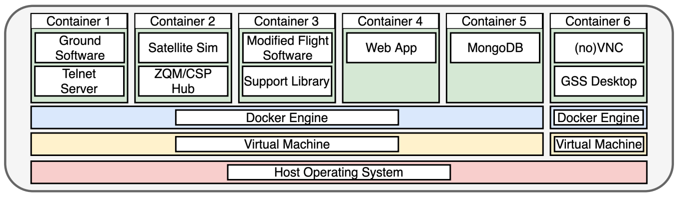


Fig. 6. Architecture of dockerized Generic CSP Honeypot

As we mentioned in Sec. II-A, 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 [83] 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 [27]. Second, it proves a convenient and flexible way to deploy HoneySat.

TABLE VII
Correspondence between real satellite mission components and HoneySat's design components.

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

Appendix B
Interaction Sequences and Exploits

Table VIII includes all the interaction sequences and exploits we performed during the experiments in Sec. VI-B.

Appendix C
Hardware-in-the-loop Experiment Results

Fig. 7 depicts a snippet of radio signal data which confirms that the hardware-in-the-loop experiment in Sec VI-F was successful. This was provided by the aerospace company.

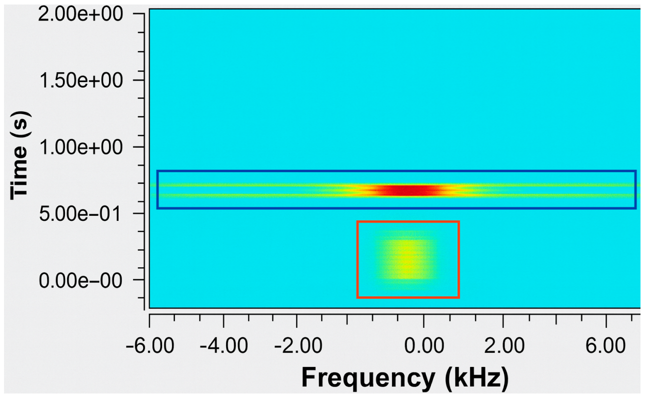


Fig. 7. Waterfall plot that shows the radio signal spectrogram when HoneySat communicated with SmallSat X. The signal on top (blue box) belongs to the telecommand sent by HoneySat, and the signal on the bottom (red box) belongs to the telemetry received from the SmallSat X.

TABLE VIII
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 vulnerabilities. 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 artifact 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.	Escape 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.	Escape 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 a malicious application that sends data over CSP
Command and Control	Telecommand a Spacecraft	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 attacker 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 parameters. 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 launch 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.

Appendix D
Artifact Appendix

This appendix accompanies the paper *"HoneySat: A Network-based Satellite Honeypot Framework"* and provides detailed instructions for obtaining, installing, and evaluating the artifact submitted for NDSS 2026 Artifact Evaluation.

A. Description & Requirements

Our artifact contains a Dockerized version of our honeypot framework, along with all the necessary components to bootstrap honeypots for satellite missions supporting two different space protocol stacks (CSP and CCSDS/YAMCS). To aid in evaluations, we also included utility scripts that facilitate the evaluation of data provided by the honeypot.

1) How to access: The artifact can be found at https://doi.org/10.5281/zenodo.17871431.

2) Hardware dependencies:

- 25GB of disk space
- at least 4 CPU cores
- 8GB RAM

3) Software dependencies: Most Linux distributions will work. We verified the artifact on Ubuntu 24.04.2 LTS.

- Docker (with compose and related components, we recommend to follow https://docs.docker.com/engine/install/)
- Python 3.12 (more recent versions may also be compatible)
- Python virtual environment
- Bash
- Telnet client
- Any modern web browser

4) Benchmarks: No External Benchmarks required.

B. Artifact Installation & Configuration

a) Extract files: Extract files from tar file downloaded from link:

```
tar xzpvf ndss-artifact-eval.tar.gz
```

b) Repository Layout: The root directory contains two subdirectories:

- **`deployment/`**: Dockerized HoneySat services with detailed setup documentation.
- **`evaluation/`**: Python utilities and scripts to help evaluate HoneySat's capabilities.

The evaluation utilities require Python 3.12 and the packages listed in `evaluation/requirements.txt`.

c) Create and Activate a Python Virtual Environment.: From the repository root:

```
python3 -m venv .
source bin/activate
```

d) Install Python Dependencies: Install the required packages for the evaluation utilities:

```
python3 -m pip install -r evaluation/
    requirements.txt
```

e) Configure Docker user: We need to add the current user to the Docker group to avoid using sudo with Docker all the time.

```
sudo usermod -aG docker $USER
sudo reboot now
```

f) Start the Honeypot Services.: All remaining setup is handled by Docker containers. You can either:

- Run the provided convenience scripts (see "Quick Start for Pass Simulations" below), *or*
- Follow the guide in **`deployment/README.md`** to bring up the services via Docker Compose.

g) Quick Start for Pass Simulations: A key HoneySat feature is its realistic communication windows: the satellite is reachable only while passing over a ground station. Many experiments benefit from a setup in which the satellite becomes reachable immediately or within a short time window. We provide two helper scripts that compute a suitable ground-station location along the satellite's predicted ground track so that a pass occurs shortly after startup. Each script builds and starts the necessary Docker Compose services and will shut them down cleanly on `CTRL+C`.

- **CSP-based honeypot:**

  ```
  ./evaluation/experiment-1/run-experiment-csp.
      sh
  ```

  This script builds and starts a CSP honeypot instance, and positions the ground station to enable communication almost immediately, allowing you to observe how an attacker would perceive a pass without waiting for a real one.
- **CCSDS/YAMCS-based honeypot:**

  ```
  ./evaluation/experiment-1/run-experiment-
      ccsds.sh
  ```

  This script builds and starts a CCSDS and YAMCS-based honeypot instance, using the same predicted ground-station placement to trigger an imminent pass.

Please proceed with the experiments described in the following sections, or consult **`deployment/README.md`** for additional configuration details.

C. Experiment Workflow

The following experiments all involve starting the honeypot system and interacting with it in some way (either programmatically or manually). The workflow typically involves issuing a startup command and then performing tasks outlined in the experiment description. For some tasks, timing matters; for example, the TCs have to be issued while the simulated satellite is reachable via the simulated ground station.

D. Major Claims

We make the following claims in our paper

- (C1): HoneySat can be used to deceive adversaries and log activities.
  - HoneySat's simulator provides believable TM (Experiment 1.1)

  - HoneySat has realistic communication windows (Experiment 1.2)
  - HoneySat provides interaction capabilities (process TC, provide TM) (experiment 1.3)
  - HoneySat Logs interaction details (Experiment 1.4)
- (C2): HONEYSAT Is extensible and supports two different protocol ecosystems
  - HoneySat is configurable (Experiment 2)

E. Evaluation

1) Experiment 1 (E1.1): [Believable TM/TC] [10 human-minutes]

Goal. Demonstrate that HoneySat produces believable telemetry (TM) in response to telecommands (TC) for a CCSDS/YAMCS setup.

Preparation. Ensure dependencies are installed as listed above.

Execution.

```
./evaluation/experiment-1/run-experiment-ccsds.
    sh
```

After the script completes and prints telemetry values, you may also inspect the telemetry via the YAMCS web interface.

Expected Results. Believable current, voltage, and temperature values for the selected battery configuration:

- Voltage: 8000 mV (normal test case)
- Temperature: 30 °C
- Current draw: 74 mA

2) Experiment 1.2 (E1.2): [Believable passes] [20 human-minutes]

Goal. Show that HoneySat enforces realistic communication windows: the satellite is reachable only during predicted passes over a ground station.

Preparation.

```
./evaluation/experiment-1/run-experiment-csp.sh
```

This positions the ground station so that a pass begins approximately 2-3 minutes after startup (adjustable), and starts all required services.

Execution.

1) Open the web interface at http://localhost:80. Log in with username `admin` and password `admin`. Click the ground-station icon and view the next predicted passes. The pass should soon show as "ongoing."
2) In a separate terminal, connect to the CSP telnet interface and activate it:

```
telnet localhost 24
# In the telnet session:
activate
```

3) Probe satellite reachability every few seconds:

```
1: com_ping 10
```

Expected Results. The satellite responds to pings only during the predicted pass and not before or after. Responses will indicate the expected addressing (source address 1, destination address 10).

3) Experiment 1.3 (E1.3): [Simulate Interaction] [15 human-minutes]

Goal. Demonstrate interactive capabilities once the satellite becomes reachable.

Preparation. Repeat the setup from Experiment 1.2. Review the command reference in the *experiment-1.3* directory.

Execution. After the satellite becomes reachable (as verified in Experiment 1.2), issue commands via the telnet interface. For example, to execute arbitrary shell commands on the OBC:

```
1: obc_system [shell command]
```

Expected Results. The experimenter can successfully execute arbitrary shell commands via `obc_system` while the satellite is reachable.

4) Experiment 1.4 (E1.4): [View logging capabilities] [5 human-minutes]

Goal. Verify that HoneySat logs interactions and relevant parameters.

Preparation. Keep a HoneySat CSP instance running (e.g., from Experiment 1.3), optionally after interacting with it.

Execution.

```
python3 ./evaluation/experiment-1/
    python_dump_mongodb/dump_mongodb.py
```

Alternatively, connect to the MongoDB instance with a database client of your choice.

Expected Results. The script prints MongoDB contents to `stdout`, showing logged parameters and interaction details.

5) Experiment 2 (E2): [Customization]

Goal. Demonstrate HoneySat's configurability across protocol stacks and scenarios.

Preparation. Navigate to *./evaluation/experiment-2*.

Execution.

1) Create a baseline customization:

```
python3 ./honeysat.py [csp|ccsds] "{
    satellite_name}" "{location}"
```

For example:

```
python3 ./honeysat.py csp "BEESAT" "Berlin"
```

2) Start services with the generated configuration (or configs in this directory):

```
python3 ./honeysat.py start [csp|ccsds] "{
    satellite_name}" "{location}"
```

For example:

```
python3 ./honeysat.py start csp "BEESAT" "
    Berlin"
```

3) When finished, stop the services:

```
python3 ./honeysat.py stop [csp|ccsds]
```

For example:

```
python3 ./honeysat.py stop csp
```

Expected Results. HoneySat can be configured with relatively low time effort to reflect different satellites, locations, and protocol ecosystems.