École Polytechnique Fédérale de Lausanne

Satellite Security
A Dive in the SatIoT Domain

by Franklyn Josef Sciberras

Master Project Report

In collaboration with CYD, Armasuisse

Johannes Willbold (Ruhr University Bochum)
Project Advisor

Martin Strohmeier (Armasuisse)
Project Co-advisor

Prof. Dr. sc. ETH Mathias Payer (EPFL)
Project Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 5, 2023

Abstract

The remarkable expansion of Internet of Things (IoT) deployments driven by recent technological advancements has highlighted the need for a truly global solution. However, existing connectivity solutions cover only a fraction of the world, particularly impacting remote areas like the Arctic due to limited coverage [8]. To address this issue, integrating satellite connectivity emerges as a promising solution, enabling comprehensive global IoT coverage. However, as accessibility to data and connectivity increases, it is crucial to address the associated security concerns.

This study provides a comprehensive analysis of the challenges and threats associated with satellite-based IoT systems. Specifically, we focus on analyzing potential threats related to the Command and Control protocol, as described in [12]. We examine the implementation of this protocol by Astrocast, a prominent Swiss vendor in the market. Our analysis considers two threat models: one where the attacker has access to public information but lacks the signing key used for Telecommands, and another where the signing key is disclosed.

Our findings demonstrate the protocol's inherent resilience to various attacks due to its simplicity. However, the disclosure of the signing key exposes vulnerabilities, enabling attackers to enumerate supported functionality and potentially exploit the system maliciously. This underscores the importance of strong keys, well-designed rotation policies, and vendor-specific key management practices. Furthermore, we highlight the future implications of a post-quantum world, where the absence of quantum-safe ciphers may impact the security of such ecosystems.

By shedding light on the challenges and threats within satellite-based IoT systems, this study contributes to the understanding of the necessary measures to ensure secure and reliable global IoT connectivity. Future research should explore the implications of post-quantum scenarios on the security of these systems.

Contents

Chapter 1

Introduction

In the words of the International Telecommunication Union (ITU), Internet of Things can be defined as "a global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, inter-operable information and communication technologies" [5]. This dogma has brought to life a way of incorporating the physical world into the digital world. The collected data via wireless sensor devices is used in a myriad of applications, by different stakeholders ranging from private users, business users, government users and the army [9].

Recent advances in technology have enabled IoT deployments to expand significantly, with ease, and at scale. Yet these connectivity solutions still provide coverage for only 15% of the world, where despite the powerful demand for a truly global IoT solution, projects in remote areas such as the Arctic are still suffering from the lack of. The addition of satellite connectivity creates a complete, blended solution to support any IoT deployment and deliver full global coverage. The latest generation of affordable Satellite IoT (SatIoT) solutions is transforming the ecosystem, allowing integrators to embed satellites within a blended IoT connectivity solution which is both global and portable [2, 3, 8]. Despite the massive amount of potential this increased connectivity and en-masse accessibility to data, one has to also factor in the increased security concerns that this poses.

Several work has been done in relation to the IoT Ecosystem where challenges have been exposed and mitigations proposed [6, 7, 11]. Yet, the adoption of satellite links widens the threat surface. The ease to eavesdrop, tampering, disrupting, and rerouting the satellite traffic provides the attacker with opportunities to launch cyber-attacks at scale and affect aforementioned stakeholders. The severity of the cited threats could be even higher when considering that most of the current satellite systems either do not integrate security at all or run outdated security techniques, unable to face the complex attacks launched today [13].

Papers such as the one by Tedeschi et. al. analyze similar attacks; attacking satellites at the

physical-layer through techniques such as jamming and spoofing [13]. As such we will omit these threats from our analysis. Instead we take inspiration from the approach described in Space Odyssey [14]. More specifically we narrow down our scope to the analysis of the protocol used by SatIoT implementations, to have bidirectional communication between a ground station and a satellite. The idea here is to evaluate whether there are some vulnerabilities in the protocol and/or implementation of this Command and Control mechanism. We believe that this is a suitable term since these ground stations have the capability of sending TCs (Telecommands) to an orbiting satellite which can have specific side effects both on the satellite itself, the IoT devices which are communicating with the satellite and/or on the information retrieved from the satellite in the form of TM (Telemetry).

In the rest of this report, we will describe the framework designed to analyze the protocol used to send such TC/TM traffic, as the recommended standard for space data systems by the CCSDS (Consultative Committee for Space Data Systems) [12]. We will proceed to apply the framework to an implementation of the standard done by one of the leading Swiss vendors in this particular market. In this analysis, we will demonstrate that the simplicity of the standard itself is sufficient to provide enough security guarantees from an external malicious actor.

We will consider two threat models. In the first one we will assume that the malicious actor doesn't have access to the signing key being used to sign TCs, but he can leverage public information through OSINT techniques (such as the vendor's protocol [1], ccsds protocol [12], and the Ground Systems and operations - TM and TC packet utilization manual by the ECSS (European Cooperation for Space Standardization) [10]). As our second threat model, we assume the same threat actor, however, this time having access to the signing key. Here we demonstrate that despite not knowing the specific implementation adopted by the targeted vendor, the TMs that the satellite responds with disclose enough information for the threat actor to map out and to some extent leverage most of the functionality of the satellite which could affect the result of the ecosystem, despite it being vendor-specific.

Chapter 2

Background

2.1 Satellite Service Context

To better understand our framework, we first go through the ecosystem in scope and how it works. As shown in Figure 2.1 we have an IoT Device Embedded with a vendor-specific communications chip. This chip will establish a connection with the satellite when it is within its orbit. Once the connection is established, the IoT device will mostly send the collected data on the uplink. However, in some cases, the satellite can be instructed to send some data to the IoT device on the downlink [1].

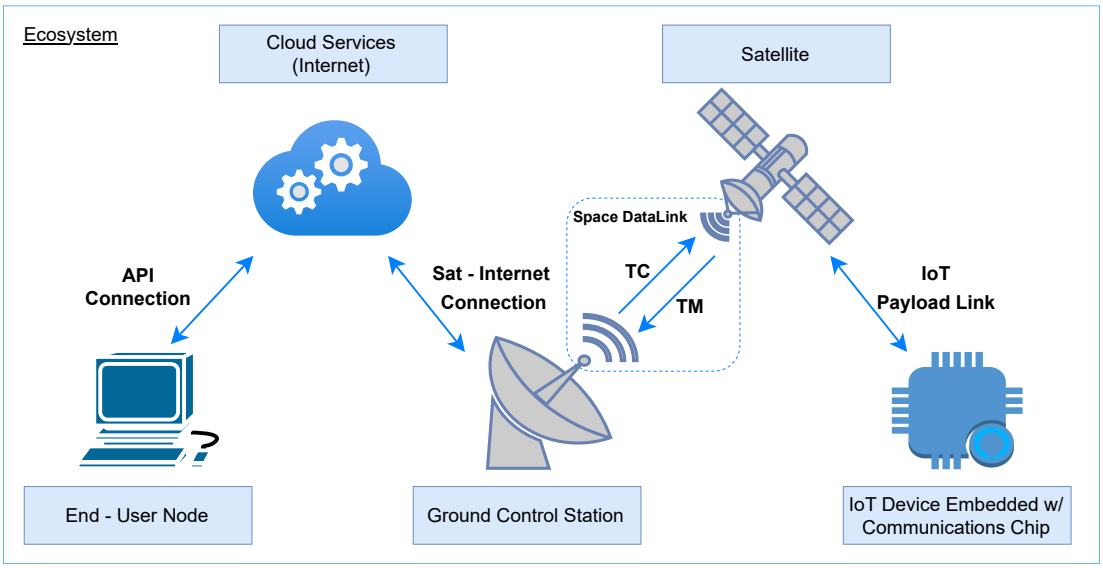

Figure 2.1: SatIoT Ecosystem

This instruction would first be generated by an End User, which through an API it would reach

some Management Service residing on the cloud. This service would have a connection with a Ground Control Station which can communicate with some Satellite in orbit. The Ground Control Station would communicate this instruction through a Telecommand (TC) also known by the satellite, sent on the uplink. The satellite itself would then send Telemetry (TM) packets on the downlink as a response to the TC received. From now on we will refer to this communication link as the *Space Data link* as described in [12]. The predominant means of satellite transmission in sending these TCs and TMs is radio and microwave signals, where higher frequencies (shorter wavelengths) are capable of transmitting more information than lower frequencies (longer wavelengths), but require more power to travel longer distances. The vendor which we are using for our case study is operating on the L-band spectrum.

As a basic outline for a satellite uplink: data enters a modem, then is sent to an up-converter, onto a high-power, whose frequency would match the reserved bandwidth. Downlinks then go from the satellite antenna to a low-noise amplifier (LNA), to the down-converter, to the modem, and then onward to a computer and/or end user. Techniques like code division multiple access (CDMA), frequency division multiple access (FDMA), and time division multiple access (TDMA) are used to increase the amount of information sent over such channels [4].

2.2 Space Protocol

In Figure 2.1 we see the utilization of a satellite communications protocol, termed as the "space protocol," to facilitate the transmission of TC/TM traffic through satellites. The primary authority responsible for establishing and disseminating such space protocols is the Consultative Committee for Space Data Systems (CCSDS), which operates as a consortium comprising multiple space agencies that collaborate to establish industry standards. The CCSDS offers a comprehensive range of protocol standards that enable effective communication among various entities and components involved in spacecraft operations. These standards encompass all layers of the OSI model and typically provide multiple options for each layer. The protocols discussed in this research encompass the Space Data Link Security (SDLS), which addresses the data link layer and incorporates a security extension, as well as the Space Data Link Protocol for the TC/TM data link [14].

2.3 Physical Layer Protection

Communications satellites usually have little to no protection. Techniques like Direct Sequence Spread Spectrum (DSSS) are used in cases such as military GPS, yet in most scenarios, if you know the right frequency, have a powerful enough transmitter and antenna, and know where to point your signal, you can hinder the communication [4]. We are aware that attacks at the physical layer such as Uplink Jamming, Downlink Jamming and Spoofing can be used against the Space Datalink.

However, in this study, we will only focus on vectors that this link could expose at the data layer.

2.4 TC Packet Data Field Header

In Figure 2.2 we start by having a look at the TC packet data field header. The first bit in this header is used to identify whether the TC packet will consist of a secondary data field. The next 3 bits are defaulted to a value of '001' by the current CCSDS standard but allow for future development. The Ack field is the first interesting field, which we could use in our study. Each of the 4 bits represents a type of acknowledgement that the ground control station would like to receive in response to the sent TC (see table 2.4). The final two important fields are the Service Type and the Service Sub-Type. The service Type defines the 256 possible TC types that the Satellite could implement. For each TC type, there exists the possibility of implementing another 256 tangible executions each performing its logic sequence. Some of these TCs are explicitly defined by the CCSDS standard, however, the majority of the space is vendor specific. One would then append the payload which complements the TC, to this header. Prep-ending the TC packet data field header would be the TC packet header. However, we will not dive into the details of this as fields within this header are there to ensure correct parsing and processing of the TC and are simplistic in nature.

Telecommand Packet Data Field Header

CCSDS Secondary Header Flag	TC Packet PUS Version Number	Ack	Service Type	Service Sub-type	Source ID	Spare
Boolean (1 bit)	Enumerated (3 bits)	Enumerated (4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Deduced	Bitstring (n bits)
					Optional	

Figure 2.2: TC Packet Data Field Header [12]

The bit settings correspond to the stages as follows:				
-	-	-	1	acknowledge acceptance of the packet
-	-	1	-	acknowledge start of execution
-	1	-	-	acknowledge progress of execution
1	-	-	-	acknowledge completion of execution

Table 2.1: Ack Flags

2.5 TM Packet Data Field Header

Similarly in Figure 2.3 we can see the TM packet data field header. The first byte is not interesting as it consists of 2 spare bits and the static PUS version number set to '001'. However, by using the

Service Type and Service Sub-Type we can identify different types of TM responses, where each unique one would carry a unique payload. One has to note that the possibility space (i.e. 256 Service Types and 256 Sub Types per Service Type) is shared with the TCs. This means that if there is a TC of Service Type 5 and Subtype 12, there can't be a TM using the same Service Type and Subtype. Once again, some of these TMs are explicitly defined by the CCSDS standard, whereas the rest are vendor-specific. The payload carrying the data from the Satellite would then be appended at the end of the TM. When it comes to the TM packet header, this has an identical structure to the one used in the TC and serves the same scope so we will not cover it here.

Telemetry Packet Data Field Header

Spare	TM Source Packet PUS Version Number	Spare	Service Type	Service Sub-type	Packet Sub-counter	Destination ID	Time	Spare
Bitstring (1 bit)	Enumerated (3 bits)	Bitstring (4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Unsigned Integer (8 bits)	Deduced	Absolute Time	BitString (n bits)
					Optional	Optional	Optional	

Figure 2.3: TM Packet Data Field Header [12]

To secure these packets the CCSDS standard recommends the implementation and inclusion of a security header within the final frame which is sent to the satellite. Despite having some guidelines on how this should be implemented, the standard leaves it quite open-ended to the vendor. In the following section, we will describe the tools we had at hand at the beginning of this project as well as the design of the new framework which will be used to analyze this particular implementation of the Space Data Link protocol, also including the reverse engineering of our vendor's implementation of the security layer at this level of the stack.

Chapter 3

Design

In this section, we will introduce the design of the framework and the motivations behind each decision. However, we first have to highlight the fact that this work complemented and was built on top of the work already done by my project advisor, Johannes Willbold. In his work, a TM parser was built which can read the raw bytes sent by satellite as a response to a TC and parse them into a structured TM with well-defined headers, data field headers and the payload. Furthermore, most of the TM packets which were described in [10] were also implemented. This gave us a good starting point since we did not have to worry about lower-layer parsing at a bit of granularity, and instead, we could directly reason at the TM level of the stack. In addition, the Swiss vendor Astrocast, was kind enough to provide us with their physical module towards which we could send the TCs that we build and it would respond with the respective TMs according to how a deployed satellite would respond, simulating the real environment. Along with the physical module, they also provided us with a script which would create the sequence of bytes corresponding to the TC generating a HouseKeepingReport, sign it and assemble the final frame as required by the vendor's specification.

The first part of our work consisted of a 2-pronged approach; i) reverse the sequence of bytes hard-coded in the given script and which is representing the HousekeepingReport TC, so that we would be able to understand whether Astrocast were conforming with the CCSDS recommendations and if so, how they built their TCs, and ii) reverse engineer the pipeline used to go from the hard-coded sequence of bytes representing the TC to the final TC which would eventually be transmitted. This is expected to consist of all the necessary layers including all the headers, recommended error check-sums as well as the security implementation needed by the satellite to be able to process the TC.

Once we had a tangible model as well as a well-defined pipeline (Figure 3.1) which produces TCs that conform with Astrocasts' specification as well as CCSDS, we could start analyzing the system under different threat models. The first threat model we will be using is that of a malicious actor who besides having access to publicly available information (OSINT) more specifically the

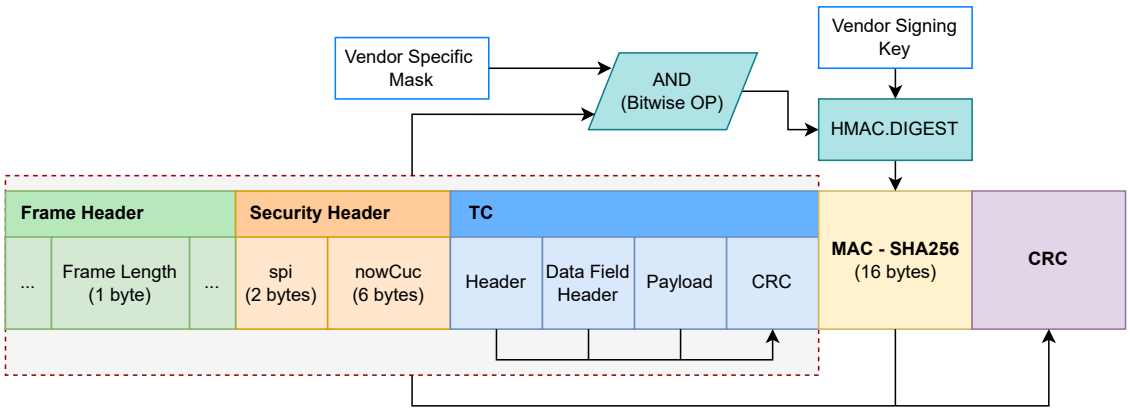

Figure 3.1: Vendor's TC Assembly Pipeline

Api of Astrocast [1], CCSDS recommendation standard [12] and the ECSSS report [10]. He also has access to the secret key used to sign the generated TCs as part of the security implementation. One should note that this is a valid threat model, since this particular vendor never provided us with information regarding the establishment and exchange of such key, nor the key rotation policy that would be used if disclosed. For these reasons, we will assume that the 256-bit key (as advertised on the vendor's page and verified in the supplied script), is hard-coded on the satellite and the ground control station. The second threat model is more restrictive and the threat actor this time would only have access to publicly accessible information including valid TCs broadcasted by authentic ground control stations.

Under the conditions of the first threat model, we would first try to enumerate which TCs the satellite supports by analyzing the TMs sent in response to the specially crafted TCs. The idea is to use the reverse-engineered TC model which we know for a fact is an accepted TC and modify it in such a way as to keep it a validly signed TC but with a different Service Type and Service Sub-Type. For these two 8-bit fields, we will enumerate the whole possibility space. Using the TM Parser available, we would then try to parse the TMs incoming as responses and see what information the satellite will leak. Since some TMs are vendor specific these will not be part of the current implementation of the TM parser. As such, we will augment it to keep track of any bytes that it was not able to parse, for further manual analysis.

However, to be able to deduct any conclusions we will need a way to correlate the sent TCs to the TMs incoming as response. Since the CCSDS recommendation does not provide any fields in its standard which would correlate a TC to the respective TM and the channel used to send a TC is different from the one on which we receive the TM (asynchronous), we designed our way of correlating the two. This consists of sending the probing TC whose response is unknown, in between two GenerateHouseKeepingResponseTCs. Since we have access to this TC we can observe the TMs generated by the satellite as a response to it. From the vendor's documentation, we know that the satellite uses a bus which means that TCs are processed in order of arrival. We can then filter out the

two expected responses and the response in between would be the TMs generated in response to our probing TC. This technique simplifies the enumeration process, where for every Service Type we are enumerating we create a fresh log, and then start iteratively enumerating for the Service Sub-Types. All the responses will then be logged and analyzed by a cleaner script which will filter out the delimiting known responses and assign an id to the group of TMs found in between every pair, which increments with every group. This id will start from 0 which would be identical to the Service Sub-type of the first TC sent to probe that particular service type, allowing us to map the generated TMs for each sent TC. A periodic TM which represents the heartbeat is also expected from the satellite. This expected to have some distinguishing structure from the rest of the TMs, which makes it very easy to filter out.

Once we have a complete list of supported TCs, we will begin mapping them out to the potential respective functionality they should be executing on the satellite. Using [10] as a reference, we will try to see if the discovered TCs with their respective Service Type and Service Sub-Type will conform with the structure detailed. One way of doing this is to first try to map the discovered TCs to the ones defined in the reference by the cited Service Type and Service Sub-Type. Then to confirm that is the same implementation we will compare the defined payload in the standard to the one accepted by the satellite for the respective discovered TC. Even though most of the specified TCs in the reference have a flexible payload structure, for the majority we can define a minimum payload length. Based on this, we will send payloads of variable length for each discovered TC and see if the response leaks any information. If we can identify the minimum required length for the particular TC accepted by the satellite and it also matches the corresponding mapped TC in the reference, we can say with some level of confidence that the identified TC is executing the described functionality, on the satellite, based on the fact that Service Type, Service Sub-Type and payload length match.

As our third approach for this threat model, we will try to see what implications the SPI used in the security header has on the TC and its execution. To do so we plan on firstly trying to enumerate the SPIs which are accepted by the satellite by iterating over the 2-byte possibility space, using the GenerateHouseKeepingReportTC as our carrier. Using the leaked information we will try to generalize any possible claims by applying the newly acquired information to all the discovered TCs.

Moving on to the second threat model, we have a big hurdle in the fact that the signing key is undisclosed. Without the signing key, it is very difficult to modify any of the final TC contents without breaking the signature, given that the pipeline is using HMACSHA256 which is a cryptographically secure signing algorithm. However, given that the satellites have a restricted amount of power they can use due to their nature, they tend to be equipped with slower processors. In our first attack under this threat model, we will try to leverage this fact by executing a timing attack. The idea is to first iterate over all the discovered TCs under the first threat model, and try sending packets signed with an incorrect signing key. Once we enumerate all the TCs that leak any information, we will narrow our scope down to them, and try to perform a classic timing attack against each one. This will reduce the complexity from 256 by 256 (bruteforce attack) to 32 by 256. Given that we have no information regarding the key rotation policy, a brute-force attack is possible if you have enough

computational resources, however, due to time and resource restrictions we will move ahead with the timing attack.

For the last part of the analysis, we will assume that the threat actor was not able to retrieve the signing key. In this scenario, what he might do is capture valid packets signed by an authentic ground station and replay them on his timings. To simulate this scenario we will first analyze the nowCuc field, which seems to be the component related to time-stamping the packet and see if there are any inherent vulnerabilities in the way this is generated. Finally, we will attempt to identify the resolution up until which the satellite accepts packets, by incrementally manipulating the aforementioned field.

Chapter 4

Implementation

In this section we will describe the most salient points of our implementation, expanding on what was described in the previous section. We start off our implementation by implementing the pipeline as described in Figure 3.1 in a modular way and using coding patterns such as the Factory Pattern. This and the use of interfaces allow us to have a dual implementation for each part of the pipeline. This would consist of one version which includes the correct implementation as described by Astrocast and its dual which is a modified implementation allowing us to only switch part of the pipeline keeping the rest authentic. After we had this in place we started sending some modified packets and we instantly notice a sequence of bytes which the TM parser was not able to parse. Having the same structure and length, a sample of them was collected and manually analyzed. This turned out to be a TM of type Verification and Subtype 2, whose structure was analogous to the TCAcceptanceReportFailure as described in [10]. After extending the parser to be able to parse this type of TM, we now could understand whether a TC was accepted or not and if not we could get a hint why, through the status code within the TM. With the leak of information that these TMs were emitting, we could start using them in our threat analysis as an oracle, where the status codes as described in table 4 [10] would allow us to infer the behaviour of the satellite in particular instances and modify our TCs and analysis accordingly.

Status Code	**Description**
0	illegal APID
1	incomplete or invalid length packet
2	incorrect checksum
3	illegal packet type
4	illegal packet subtype
5	illegal or inconsistent application data
>5	mission specific codes (usually packet is accepted)

Table 4.1: TCAcceptanceReportFailure - TM Status Codes

Algorithm 1 shows our main driver for the threat analysis under different threat models. The driver will establish a connection to the satellite and initialise a queue. These in turn will be passed to a reader thread which will simulate the downlink as well as the threatModel which will simulate the uplink. As we mentioned in the Design section, these two channels are independent of each other. This is where the queue becomes useful. The reader thread started as a background process, whose job is to read any TMs broadcasted by the satellite. It will then use the *parsingFunc* to parse the TMs by filtering out the known responses for the GenerateHouseKeepingReportTC and whilst doing so, create association ids which are used to label any responses found in between the two expected TMs. The labelled responses are then sent over the queue, where the threatModel function is also listening. Using such labels the threatModel can proceed to map the responses to any TCs initially sent for the particular type of analysis, it is covering.

Algorithm 1 Driver Function for Threat Models

```
1: q = Queue()
2: con = syncNewConnection()
3: reader = startReaderThread(con, q, parsingFunc)
4: threatModel(con, q)
5: reader.terminate()
6: con.terminate()
7: q.clean().terminate()
```

In Algorithm 2 we show the implementation of our first threat model. Over here we iterate over the serviceId space which is of size 8 bits. For each service, we first send a probing TC, which is a TC mirroring the GenerateHousekeepingReportTC but instead, it has the serviceId we are probing, set in the Service Id header and Service Sub-Type set to 0 (Figure 2.2). From table 4 we now know that if the response has a status code of 3, the service type is not supported. If we had to brute force over the whole space (Service Type: 8bits, Service Sub-Type: 8bits) this would take 256256=65,536 iterations. Keeping in mind that the processor on the satellite is low power so it takes some time to process each TC, and given the fact that we pace our TCs with a couple of seconds of delay per iteration such as to not overwhelm it, this enumeration process would take an exorbitant amount of time.

Using the status code as an oracle which allows us to skip iterating over the 256 Service Sub Types for services which are not supported, dramatically reduces the computation time to (1+n)*256 iterations where n is equal to the number of Service Types supported and thus is expected to be n « 256. On the other hand, if the status code is anything else, it deserves further manual investigation since it most probably is supported. As such we create a log file for the service and iterate over all possible subtypes, where the parsed responses are logged for manual inspection.

From manual inspection, we observed that whenever the Service Type is supported but the Service Sub-Type included in the TC is not, we were getting back a status code of 4, which is as expected and as such these TCs would be discarded. Supported TCs, where both Service Type

Algorithm 2 Threat Model: Service Enumeration

```
1: Inputs: con: Connection, q: Queue
2: for serviceId in range(0,255) do:
3:     probeService(con, serviceId)
4:     response = q.get(block=true)
5:     if response.statusCode == 3 then
6:         skip()
7:     else
8:         logger = createLog(serviceId)
9:         for serviceSubType in range(0,255) do:
10:            frame = astrocastPipeline(serviceId, serviceSubType)
11:            sendTC(con, frame)
12:            response = q.get(block=true)
13:            logger.log(response)
```

and Service Sub-Type are correct, would return either a random status code (mission specific and vendor-based), which would mean that the payload sent was of the correct length but the contents were incorrect, or else the status code 5, which meant that the payload sent was of incorrect length. In Algorithm 3 we leverage this information to find the minimum payload length expected from each discovered TC, which ultimately also helped us during the manual process of mapping out functionality for them. During the implementation and testing of this ThreatModel, some observations were made. More specifically it seems that this particular implementation expects at least one byte of null data in the payload, even when the TC's functionality would not need it (for example Enable/Disable some onboard service). Deriving from this, only payloads with an odd length are accepted, as a payload of even length would make the frame of odd length and would return a TM of type EventReportingErrorAnomalyLowSev. Finally, once the minimum payload length is identified, payloads of that length or any increment to them by two bytes would be processed by the satellite and return a TCAcceptanceReport.

Once we have enough information about the accepted TCs by the satellite, we can go over the last threat model under these threat analysis conditions. The SPI field in the security header is represented by 2 bytes which as we've already seen gives a large possibility space. Despite this, we try to first enumerate the SPIs for which the satellite would respond, besides the value of 2 which was reverse-engineered alongside the rest of the pipeline. This analysis is done using the GenerateHousekeepingReportTC. Extending the idea to the other discovered TCs, we iterate over them and for each we use a random SPI from the possibility space. This would allow us to deduce which SPIs are supported and if supported SPIs vary per TC type. One should note that we use the random SPI approach when probing the discovered TCs since we deemed it to be the next best approach given that brute force is infeasible due to time restrictions.

Next, we restrict our threat model to an attacker who does not have access to the secret key. For our first threat analysis, we will try to evaluate whether a timing attack on the secret key is possible,

Algorithm 3 Threat Model: Payload Length Analyzer

```
1: Inputs: con: Connection, q: Queue
2: mapLength = {}
3: for tc in DiscoveredTCs do:
4:     payload = b'\x00'
5:     done = false
6:     while not done do:
7:         frame = astrocastPipeline(tc, payload)
8:         sendTC(con, frame)
9:         response = q.get(block=true)
10:        if response.statusCode == 5 then
11:            payload += b'\x00\x00'
12:        else if response.statusCode > 5 or response.status == Accepted then
13:            mapLength[tc] = length(payload)
14:            done = true
15: mapLength.persist()
```

leveraging the slow processors traditionally used in satellites. To conduct this threat analysis, we use the GenerateHousekeepingReportTc but we amend the reversed pipeline to utilize 32 null bytes as the signing key. This TC is sent and the time it takes to receive a response is measured, where the average time is calculated. After, the first byte is incremented by 1 bit and the process is repeated. This is done for all 256 bits per byte position. At this point, the measured average times are compared and the 256-bit value which induced the longest response time is chosen and fixed for that particular byte position. Repeating this process for all 32 bytes we should be able to leak the signature key. Once again a brute force approach is also possible here, however, this approach has a cheaper computation time of $32 * 256 = 8{,}192$ iterations.

As our final threat model, we reference algorithm 4. In this threat analysis, we emulate a replay attack of some pre-recorded packets by an attacker, where the signature is valid but the contents of the packet cannot be modified. To evaluate how resilient such a system is against these kinds of attacks we try to find the time resolution up to which the satellite would accept a packet. From this, we should be able to deduce whether an attack of this nature is tangibly feasible or not. As seen in algorithm 4, after coarsely testing different time resolutions by hand we narrow down the time spectrum to 4 minutes. Using a model which would also allow for futuristic attacks, we split the time window in two and use a 10-second time step. For each time delta we generate a relative timestamp which we then use to construct a valid TC, in doing so simulating recorded packets. Each TC is sent and the response of whether it was accepted or not by the satellite is recorded.

Algorithm 4 Threat Model: Replay Attack

```
1: Inputs: con: Connection, q: Queue
2: acceptedResolution = []
3: timeDelta = [−2mins : 10secs : +2mins]
4: for delta in timeDelta do:
5:     currentDateTime = DateTime.UtcNow()
6:     newDateTime = currentDateTime + timeDelta
7:     frame = astrocastPipeline(tc, payload)
8:     sendTC(con, frame)
9:     response = q.get(block=true)
10:    if response.status == Accepted then
11:        acceptedResolution.append(delta)
12: acceptedResolution.persist()
```

Chapter 5

Evaluation

In this section, we will evaluate the results and observations made under each previously described threat model. More specifically we start by observing Table 5.1. In this table, we have an enumerated list of some of the TCs discovered during the enumeration process. More specifically, this is a subset of TCs whose respective minimum payload we were also able to identify. This helped us match the TCs identified Type and SubType to a matching description. However, one should note that some of these TCs are Astrocast-specific and are labelled as Unknown. The reason for this is that to be able to identify what these TCs are doing, we would need to be able to specifically analyze the side effects that each of these commands would have, and then reverse engineer the ongoing process. Such a task is a feat which with the current information and resources at hand was difficult to achieve. One should also highlight the importance of the Astrocast Anomalous Service 132, which seemed to demonstrate anomalous behaviour when it comes to processing the sent TCs. Tangibly this anomalous behaviour consisted of the satellite both responding with an Acceptance message as well as a Rejection message up until a certain service sub-type and after which it would altogether stop responding to the TCs for some non-deterministic amount of time. When it comes to the identified minimum payload length, it is important to state that the satellite seemed to accept payloads which have at least the specified minimum length. We utilize the word minimum, as the satellite seems to also accept any other payload whose length satisfies the criteria $minLength + 2bytes$; which seems to be in line with the fact that the total TC's length (including headers) should always be of even value.

Table 5.1: Enumerated Services

Supported Services			
Type	SubType	Inferred Description	Min. Length (bytes)
3	6	Disable House Keeping Parameter Report Generation	3
3	5	Enable House Keeping Parameter Report Generation	3
3	129	House Keeping Diagnostic Request	3
3	131	House Keeping Unknown 131	5

3	132	House Keeping Unknown 132	7
3	141	House Keeping Unknown 141	1
5	5	Enable Event Report Generation	3
5	6	Disable Event Report Generation	3
6	2	Load Memory Using Absolute Addresses	9
6	5	Dump Memory Using Absolute Addresses	9
6	9	Check Memory Using Absolute Addresses	9
6	128	Memory Management Unknown 128	7
6	129	Memory Management Unknown 129	7
6	130	Memory Management Unknown 130	9
6	131	Memory Management Unknown 131	7
6	132	Memory Management Unknown 132	7
6	133	Memory Management Unknown 133	7
6	134	Memory Management Unknown 134	7
6	135	Memory Management Unknown 135	9
6	136	Memory Management Unknown 136	9
6	140	Memory Management Unknown 140	11
6	150	Memory Management Unknown 150	1
6	151	Memory Management Unknown 151	1
11	2	Disable Release of TCs	1
11	1	Enable Release of TCs	1
11	6	Delete TCs Over Time Period	1
11	4	Insert TC in Command Schedule	1
11	5	Delete TCs	1
11	7	Time Shift Select TCs	1
11	8	Time Shift Select TCs Over Time Period	1
11	15	Time Shift All TCs	1
13	6	Repeat Parts	7
15	2	Disable Storage in Packet Stores	3
15	1	Enable Storage in Packet Stores	3
15	7	Downlink Packet Store Contents for Packet Range	3
15	9	Downlink Packet Store Contents for Time Period	3
15	10	Delete Packet Store Contents Up To Specified Packets	3
15	11	Delete Packet Store Contents Up To Specified Storage Time	9
15	12	Report Catalogues for Selected Packet Stores	3
15	17	On Board Storage and Retrieval Unknown 17	1
15	128	On Board Storage and Retrieval Unknown 128	3
19	5	Disable Actions	3
19	4	Enable Actions	3
132	*	Astrocast Anomalous Service 132	1
133	1	Astrocast Service 133 Unknown 1	3
133	2	Astrocast Service 133 Unknown 2	1
133	4	Astrocast Service 133 Unknown 4	1
133	10	Astrocast Service 133 Unknown 10	1
140	1	Astrocast Service 140 Unknown 1	65
140	2	Astrocast Service 140 Unknown 2	81
140	3	Astrocast Service 140 Unknown 3	19

170	1	Astrocast Service 170 Unknown 1	9
170	2	Astrocast Service 170 Unknown 2	1
170	5	Astrocast Service 170 Unknown 5	7
170	7	Astrocast Service 170 Unknown 7	1
170	8	Astrocast Service 170 Unknown 8	1
170	9	Astrocast Service 170 Unknown 9	1
170	10	Astrocast Service 170 Unknown 10	1
170	20	Astrocast Service 170 Unknown 20	9
170	21	Astrocast Service 170 Unknown 21	1
170	128	Astrocast Service 170 Unknown 128	3
3	128	Generate House Keeping Report	3

Table 5 is an extension of the previous table where it contains TCs discovered during the enumeration process, but whose minimum length we were not able to uncover. The reason behind this is that the satellite kept on rejecting any byte string of null bytes which we would send to it up until the payload length field would be exhausted. In this case we are not sure if the TCs in question are missing a minimum payload length check in their implementation on the satellite or else some other misconfiguration which could lead to deeper problems. For the reason of time restrictions, this route was not explored further.

Problematic Services		
Type	SubType	Inferred Description
170	3	Astrocast Service 170 Unknown 3
170	4	Astrocast Service 170 Unknown 4
170	6	Astrocast Service 170 Unknown 6
13	8	Abort Down Link
12	7	Modify Parameter Checking Information
12	128	Monitoring Unknown 128

Table 5.2: Services with unidentified min. payload lengths

In Table 5 we find the last set of TCs which were discovered. These services do not require a payload as part of the TC and would work as is, making it easy for us to see their side effects since we did not have to figure out the correct payload to send within the command. An interesting discovery made during this step was the discovery of Service Type 17 Sub Type 139, which would put the satellite in an unknown state permanently. By unknown state, we mean that the satellite would altogether stop responding to our TCs whilst also stop sending heartbeat messages as it would do under normal circumstances.

As the last part of the first threat model, we tried to observe if any of the discovered TCs would respond to a random SPI as well as try to enumerate the SPI field using the GenerateHousekeepingReportTC. The first approach unfortunately did not result in any effective results as the satellite did not respond to any of the TCs when a random SPI was utilized. The second approach was stopped halfway because of time restrictions, however, the trend was demonstrating similar results, where the satellite did not respond to any of the GenerateHousekeepingReportTCs using SPIs in the range of $[0,3135]$, except for SPI 2, which we knew about from the reversed pipeline.

A similar outcome was observed for the first part of the second threat model, where breaking the signature key using a timing attack seems to be very hard. This is because the satellite does not send any type of response to any TC which has an unknown signature. Given that the sent TC is absorbed, the round trip time of the response cannot be measured. Having said that, we still claim that a brute force attack would still be possible, having enough time and

Empty Payload Services		
Type	SubType	Inferred Description
11	3	Reset Command Schedule
11	17	Report Command Schedule in Summary Form
12	8	Report Current Monitoring List
17	1	Perform Connection Test
17	129	Perform Unknown Test
17	139	Put Satellite in Unknown State
19	6	Report Event Action List

Table 5.3: Services with an empty payload

resources. Especially, given the fact that key rotation policies and key management policies are not well defined in the information disclosed to us whilst researching this particular system.

Finally, we claim that a replay attack would also be very difficult to conduct against this particular system as our results show that the satellites accept TCs which are within the time frame of $[-60seconds, +60seconds]$ from the current time. This would imply that an attacker would have a very short time frame in which he could conduct such an attack, thus limiting the potential damage achievable.

Chapter 6

Conclusion

In this report, we focused on analyzing and evaluating the Command and Control mechanism used in IoT systems within a SATCOM ecosystem. The specific protocol for bidirectional communication between a ground station and a satellite was examined to identify potential weaknesses, both in the protocol as well as its implementation. To do so we design a framework for evaluating the TC Space Data Link Protocol and apply it to a Swiss vendor's implementation. Two threat models are considered, one where the malicious actor lacks access to the signing key but can leverage public information, and another where the actor has access to the signing key.

Overall, the analysis reveals that despite not knowing the vendor's specific implementation, knowing the signing key, a threat actor can gather enough information from the satellite's responses to map out most of the satellite's functionality. This puts him in a good position to exploit the entire ecosystem. However, things become increasingly more complicated if such a key is not disclosed, as the protocol they rely on is quite simple and as such it's very hard to implement incorrectly. This puts further emphasis on the usage of cryptographically sound functions such as HMAC-SHA256 and an adequate key length since the system's safety is highly dependent on their soundness. Furthermore, one has to see what are the implications of a post-quantum era on these systems, given the lack of quantum-safe functions.

Bibliography

[1] *Astrocast Docs*. URL: https://docs.astrocast.com/.

[2] *Expanding IoT deployments with cost-effective Satellite IoT*. URL: https://www.astrocast.com/news/expanding-iot-deployments-with-satellite-iot/.

[3] Xinran Fang, Wei Feng, Te Wei, Yunfei Chen, Ning Ge, and Cheng-Xiang Wang. "5G Embraces Satellites for 6G Ubiquitous IoT: Basic Models for Integrated Satellite Terrestrial Networks". In: *IEEE Internet of Things Journal* 8.18 (2021), pp. 14399–14417. DOI: 10.1109/JIOT.2021.3068596.

[4] Adam Ali.Zare hudaib. "Satellite Network Hacking and Security Analysis". In: *IJCSS* 10 (1 2016).

[5] *ITU-T Rec. Y.2060 (06/2012) overview of the internet of things*. URL: https://handle.itu.int/11.1002/1000/11559-en?locatt=format:pdf&auth.

[6] Mohammad Zubair Khan, Omar H. Alhazmi, Muhammad Awais Javed, Hamza Ghandorh, and Khalid S. Aloufi. "Reliable Internet of Things: Challenges and Future Trends". In: *Electronics* 10.19 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192377. URL: https://www.mdpi.com/2079-9292/10/19/2377.

[7] Renjith P N and Ramesh K. "Trust based Security framework for IoT data". In: *2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*. 2020, pp. 1–5. DOI: 10.1109/ICCCSP49186.2020.9315282.

[8] David Palma and Roger Birkeland. "Enabling the Internet of Arctic Things With Freely-Drifting Small-Satellite Swarms". In: *IEEE Access* 6 (2018), pp. 71435–71443. DOI: 10.1109/ACCESS.2018.2881088.

[9] Zhicheng Qu, Gengxin Zhang, Haotong Cao, and Jidong Xie. "LEO Satellite Constellation for Internet of Things". In: *IEEE Access* 5 (2017), pp. 18391–18401. DOI: 10.1109/ACCESS.2017.2735988.

[10] ECSS Secretariat. *ECSS-E-70-41A: Ground systems and operations — Telemetry and telecommand packet utilization*.

[11] A. Shobanadevi, G. Maragathm, S. Gangadharan, M. Soni, R. Kumar, T. Tran, and B. Sing. “Internet of Things-Based Data Hiding Scheme for Wireless Communication”. In: (). URL: https://doi.org/10.1155/2022/6997190.

[12] *TC Space Data Link Protocol*. URL: https://public.ccsds.org/Pubs/232x0b4.pdf.

[13] Pietro Tedeschi, Savio Sciancalepore, and Roberto Di Pietro. “Satellite-based communications security: A survey of threats, solutions, and research challenges”. In: *Computer Networks* 216 (2022), p. 109246. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2022.109246. URL: https://www.sciencedirect.com/science/article/pii/S138912862200319X.

[14] J. Willbold, M. Schloegel, M. Vogele, M. Gerhardt, T. Holz, and A. Abbasi. “Space Odyssey: An Experimental Software Security Analysis of Satellites”. In: (). URL: https://publications.cispa.saarland/3934/1/SatSec-Oakland22.pdf.