WannaFly: An Approach to Satellite Ransomware

Conference Paper · July 2023

DOI: 10.1109/SMC-IT56444.2023.00018

3 authors:

Gregory Falco
Cornell University
74 PUBLICATIONS **823** CITATIONS
SEE PROFILE

Rajiv Thummala
Cornell University
4 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Arpit Kubadia
Johns Hopkins University
3 PUBLICATIONS **26** CITATIONS
SEE PROFILE

WannaFly: An Approach to Satellite Ransomware

1st Gregory Falco
Sibley School of Mech. & Aerospace Engineering
Cornell University
Ithaca, USA
gfalco@cornell.edu

2nd Rajiv Thummala
Department of Cybersecurity
The Pennsylvania State University
State College, USA
rkt5192@psu.edu

3rd Arpit Kubadia
Institute for Assured Autonomy
Johns Hopkins University
Baltimore, USA
akubadi1@jhu.edu

***Abstract*—Ransomware is a prevailing concern across sectors, wreaking havoc on operations and causing cascading failures - often cyber-physical in nature. The space sector has experienced ransomware attacks; however, the space vehicle has not been the target of these publicly disclosed incidents. A ransomware attack against a space vehicle would need to be carefully crafted to mitigate the risk of destroying the underlying functionality of the spacecraft, while still achieving its purpose - denying access until a ransom is paid. Through static code analysis, this paper proposes an approach for deploying a ransom attack against a space vehicle that engages NASA's core Flight System (cFS).**

***Index Terms*—space cybersecurity, ransomware, satellite ransomware, satellite attack, satellite hijack, NASA, flight software, core Flight System, flight software security, space ransomware**

I. Introduction

Ransomware, a cyber attack where a threat actor denies service or access to a system until a payment is made, can have devastating impacts on a digital asset, an organization, a network of organizations, or even a global supply chain. While threat actors can wield ransomware as a blunt-force instrument with wide-spread consequences, it could also be engaged as a finely-tuned cyber payload whose impact is limited to strategic targets. Ransomware typically takes the form of an encryption malware deployed on a file system, where the decryption key is not furnished until the victim pays the attacker (usually in Bitcoin); however, ransomware could refer to a range of malware deployed to deny service. Ransomware may have different implications for different sectors, in all instances it halts operations until concessions are made by the victim. This is especially detrimental to sectors with time-specific processes such as critical infrastructure and sectors involving cyber-physical systems. The consequences of a ransomware attack would be particularly devastating against a space vehicle given both the direct impact on the space asset, which is finely-tuned, and its cascading consequences for the space vehicle's users, who rely on a satellite's services.

Space systems are increasingly targeted by cyber attackers and space vehicle security has emerged as a topic of international importance [1], [2]. Space operators are intimately familiar with ransomware threats. However, the target of ransom attacks against space missions have been the ground segment and mission operations centers, which holds the greatest attack surface. It is arguable that a ransom attack against a ground station is more efficacious in compelling a victim to pay an attacker given a single ground station or operations center generally manages multiple satellite missions - making a stronger value proposition for a victim to pay. However, financial reward is not the only motivation for a ransom attack. For example, some nation states seek to engage ransomware to cause chaos and for power signaling purposes [3]. Should there be a critical satellite mission of national importance - be that for public interest or intelligence purposes, an attacker may seek a high-profile target such as a space vehicle to make a statement.

Questions have emerged, especially at industry conferences about the viability of a ransom attack directly affecting a satellite in orbit, hence the motivation for this research. A blunt-force implementation of a ransom attack on a space vehicle could permanently disable the asset, making recovery after payment an impossibility.

This research seeks to evaluate a strategic approach to hijacking a space vehicle without compromising the link, ground or user segements, while still enabling the space vehicle's continued operations and providing a pathway to return access to the operator after a ransom is paid. This is achieved via static code analysis of NASA's core Flight System (cFS) to evaluate a malware target. Ultimately, the software bus application programming interface (API) which enables interprocess communication (IPC) was identified as the prime target for such an attack.

II. Prior Art

A. Notable Ransomware Attacks

Encrypting critical components of an asset has long been a standard method of denial of service for cyber criminals. In contrast to deploying taxing malware payloads, ransomware offers adversaries an inexpensive yet potent method of disabling a system. Dependent on the attack vector and particular strain, it bears the capability to render a system completely inoperable, leaving limited methods of recovery. The menace of ransomware continues to escalate as encryption algorithms utilized in attacks have become nearly impregnable. Concurrently, pseudo-annonymous and secure mediums to demand subsequent ransoms have significantly developed due to the advent of blockchain technology and the ubiquity of cryptocurrencies. These factors have led to a mass uptake in the frequency and notoriety of ransomware attacks.

Pertinent to this study, an examination of major ransomware attacks and their respective economic implications will provide insight into the potential damage a ransomware attack on a space vehicle can cause. We consequently focus on recent attacks to provide an accurate depiction of the current threat landscape.

The Colonial Pipeline ransomware attack was the most devastating ransomware attack recorded in American history [4]. Headquartered in Alpharetta, Georgia, the Colonial Pipeline is one of the largest pipeline operators in the United States and provides nearly half of the East Coast's fuel, including gasoline, diesel, home heating oil, jet fuel, and military supplies [5]. On May 6th, 2021, DarkSide—a hacking group believed to be operating out of Russia—deployed ransomware on the Colonial Pipeline's IT and billing infrastructure by exploiting a legacy virtual private network (VPN) account. This prompted pipeline operators to halt its operation out of concerns of the malware propagating to the operational technology (OT) controlling the physical pipeline. DarkSide demanded 75 BTC ($4.4 million equivalent) as a ransom to eradicate the malware, to which Colonial Pipeline's manager eventually obliged [6]. While normal operations resumed nearly a week later, millions in damage had been incurred. The shutdown caused price increases, gas shortages, and panic buying. DarkSide had also exfiltrated nearly 100 GB of proprietary data from the attack, forcing Colonial Pipeline to allocate additional resources for recovery [7].

JBS, a Brazilian company that supplies a fifth of the world's meat, was attacked with ransomware on May 30th, 2021 [8]. Key servers that enabled information technology equipment in North America and Australia were compromised, leading to JBS shutting down all nine of their meatpacking plants in the U.S [9], [10]. The attackers, alleged to be Russian hacking group "REvil" by the FBI, demanded $11 million in BTC as ransom, to which JBS reluctantly obliged to expedite the recovery process [11]. Servers were eventually restored and normal operations resumed, however, the attack caused significant disruptions to the supply chain, setbacks of millions of dollars, and forever tainted JBS' reputation.

The WannaCry ransomware strain is one of the most notorious computer worms in history. In May of 2017, hackers leveraged leaked tools developed by the U.S. government to exploit vulnerabilities within the Windows OS. An estimated 300,000 computers in 150 nations were infected by the self-replicating ransomware [12]. 7,000 computers were attacked within the first hour and 110,000 distinct IP addresses were compromised in just two days [13]. Primarily deployed through phishing campaigns, WannaCry targeted a wide range of victims including the National Health Service (NHS) [14], car manufacturers [15], and many small businesses. The hackers demanded ransoms up to $600 in BTC, many of which were paid by the victims. Total damage from WannaCry is estimated to be upwards of $4 billion dollars.

The NotPeyta ransomware attack is dubbed by many as the most devastating cyberattack to date. As a successor to the notorious and destructive Peyta ransomware strain, NotPeyta featured advanced propagation mechanisms which enabled rapid infection of a large number of systems. Despite Ukraine being NotPeyta's primary target, it dispersed to more than 60 countries and destroyed the computers of thousands [16]. One particular target was the global transport and logistics giant Maeserk, to which NotPeyta destroyed all end-user devices including 49,000 laptops and print capability [16]. The damage caused by NotPeyta has been estimated to be more than $10 billion [16]. It is alleged that NotPeyta had been modified from Peyta to make it technically impossible to recover the victim's files, regardless of whether they paid the ransom. Senior cybersecurity officials in the U.S. government analogized the deployment of NotPeyta to using a nuclear bomb to achieve a small tactical victory [17].

CNA Financial is the seventh-largest commercial insurance firm in the U.S. and was the target of a ransomware attack on March 21st, 2021 [18]. Hackers deployed Phoenix CryptoLocker ransomware on the CNA enterprise, which encrypted 15,000 devices including key CNA systems [19]. The strain was a spin-off of another malware known as "Hades", created by the Russian hacking organization "Evil Corp" [20]. To address the incident, CNA contacted experts and law enforcement, both of which conducted investigations into the attack. However, after a week of unsuccessful recovery efforts, CNA began negotiating with the hackers. Eventually, a $40 million ransom was paid by CNA to regain access to their systems [20], [21]. The payout was one of the largest in American ransomware history and raised alarms to the dangers of cooperating with cyber criminals [21].

The aforementioned incidents are just some of the numerous ransomware attacks that have occurred in recent times. In the first half of 2022, there were 236.1 million ransomware attacks reported worldwide and a total of 623.3 million reported in 2021 [22]. While not all of these attacks were successful, it highlights the prevalence of this attack and the danger it poses to critical systems. Failure to implement robust defenses against ransomware in critical infrastructure, extensively used systems, and emerging disruptive technologies will pose serious threats as ransomware attacks continue to increase in occurrence and potency.

B. Ransomware as a Service

In recent years, there has been an extensive adoption of the Ransomware-as-a-Service (RaaS) business model, in which ransomware developers operate a subscription service that leases ransomware kits to attackers in exchange for a percentage of earnings. Modeled after the traditional Software-as-a-Service (SaaS) paradigm, RaaS has enabled the layman or 'script-kiddie' to conduct ransomware attacks without the prerequisite of technical competence. As a result, the frequency of ransomware attacks has skyrocketed. Various methods exist to obtain RaaS products. The standard procedure is to purchase kits through the Tor browser, which is an application that enables anonymous internet browsing and communication. Web addresses stored on the Tor browser are not indexed by conventional search engines and all traffic is encrypted, providing an elevated degree of anonymity and security to RaaS developers.

In the context of this study, RaaS developers can ultimately faciliate the scalable deployment of ransomware for space vehicles given its technical feasibility as demonstrated in this study. Therefore, it is critical to expose this concern to the space vehicle community, given it is an inevitability that ransomware kits tailored to breach space vehicles will become a standard product offered by RaaS developers. Failure to account for this impending crisis in advance will pose significant risk to national security as space vehicles continue to integrate with critical infrastructure.

C. Embedded System Ransomware

Embedded systems are often fundamental computing components of critical infrastructure, enabling intelligent and efficient control over a variety of mechanical and electrical processes. Embedded systems are notoriously difficult to secure given their resource constraints, making them particularly vulnerable to ransomware attacks. Further, embedded systems often engage commercial-of-the-shelf (COTS) hardware and software components. COTS systems are readily available for attackers to study and develop exploits for, increasing their likelihood of being a target. Given that many space vehicles such as CubeSats increasingly engage COTS systems in order to reduce costs and enable interoperability, cybe risks associated with COTS embedded devices now directly impact the space vehicle.

There has been extensive efforts in recent years to raise awareness to the threat of ransomware to embedded systems. A report by [23] claimed that existing vulnerabilities identified in modern electric grids could be abused to facilitate ransomware payload deployments against its embedded systems. Others have also pointed to the technical feasibility of deploying ransomware on embedded systems. For instance, DefCon 24 featured a successful demonstration of a ransomware attack against a smart thermostat. The firmware of the device was successfully encrypted, locking users out of the device barring a 1 BTC payment [24]. [25] also featured a proof of concept ransomware capable of attacking exploitable Linux-based IoT devices.

D. Attacks on Space Assets during the Ukraine/Russia War

There have been several confirmed and alleged cyber attacks against space systems over the course of the 2022 Russia/Ukraine war. Most notably, Russia targeted Viasat at the dawn of the conflict, successfully disabling satellite communications [26]. This incident has been followed by other alleged attacks. For example, a member of ”GhostSec”–a subsidiary of the Anonymous hacktivist group–shared a tweet on March 14th, 2023, which featured screenshots indicative of a breached Russian GNSS receiver. Cybersecurity analysts at Cyble highlighted the potential validity of these claims, sharing research that GNSS-3 had a total of 899 internet-exposed instances with Russia suffering the highest number of exposures [27]. Another pro-Ukraine hacktivist organization–Team OneFist–reported on October 10, 2022, that they had successfully executed ”Operation Cataclysm”, which sought to attack ground-stations in Moscow to disable access to the satellite network of MegaFon (leading mobile operator in Russia) [28].

1) Space System Ransomware Attacks: While not confirmed by the victim in either case (which is not uncommon due to the reputational damage associated with breaches) there have been reports of ransomware waged against space operators that are concerned parties of the latest geopolitical conflict. For example, reports emerged on March 14th, 2023 that the ransomware gang ”Lockbit” had exfiltrated 3,000 confidential schematics developed by SpaceX through a breach of Maximum Industries–one of their third-party manufacturers [29]. Lockbit demanded an extortion payment from SpaceX, threatening a dox of the proprietary schematics if the ransom was not payed. Neither SpaceX nor Maximum Industries publicly confirmed being impacted by the ransomware attack.

Another instance of ransomware waged against a space operator occurred in March 2022 in which Network Battalion 65 (NB65)–a subset of the Anonymous hacktivist organization–claimed to have shut down the control center of Russia’s primary space agency (ROSCOSMOS) in response to the Russia-Ukraine crisis [30]. The director general of ROSCOSMOS denied the claims, stating that their space activity control centers were operating normally [31]. NB65 subsequently posted images to their Twitter account of a breached WS02 integrator console and what appeared to be access to a ROSCOSMOS vehicle monitoring system [32]. Based on our analysis of NB65 primary sources, the likely attack procedure that NB65 took is as follows:

1) NB65 obtained intel of the WS02 Enterprise Integrator being utilized in the Roscosmos control station.
2) NB65 exploited a log4j vulnerability in the Roscosmos WS02 Enterprise Integrator to deploy a modified version of the Russian Conti Ransomware.
3) NB65 conducted a double extortion attack to threaten dox in the event the ransom was not paid.
4) NB65 deleted various files across the server.

E. Sensitivity of Space Vehicles

In stark comparison to conventional cyber-physical, industrial control systems, space vehicles are relatively fragile and sensitive in nature. The space vehicle’s operating requirements in such extreme environments is the source of many of its sub-optimal design constraints. For instance, the harsh conditions that space vehicles must endure (high energy radiation, ultra-high vacuum, etc.) prevents the ability to rely on external systems for support. Space vehicles consequently depend on internal subsystems that work in tandem to provide a robust self-sufficient architecture capable of performing critical functions. Due to the inter-dependent nature of subsystems, one minor bug or malfunction to a singular subsystem can cause

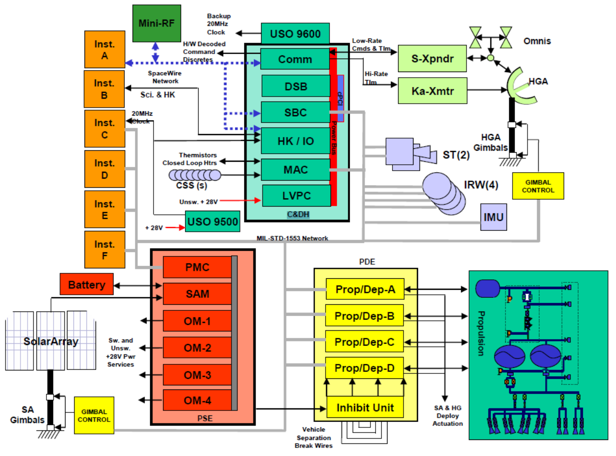

Fig. 1. Simplified Space Vehicle Avionics Schematic (NASA) [33]

mission failure. For example, figure 1 features a simplified space vehicle avionics schematic developed by NASA. In this configuration, the power bus is directly dependent on the Command and Data Handling subsystem (C&DH) to perform its functions. By extension, if the C&DH suffered a breach there would be a cascading failure to critical components of the software bus, such as the low voltage power converter (LVPC). This could ultimately result in a complete failure of the space vehicle and its mission.

There is an extensive record of such bugs, in which a seemingly trivial malfunction caused total mission failure. One example is the Mars Global Surveyor (MGS) which was launched on November 7th of 1996 on a one-year mission to study the Martian surface from a low altitude orbit. A software update pushed in June 2006 caused data to be written to the incorrect memory address, triggering a memory fault [34]. As a result, MGS reoriented to an angle that exposed one of two batteries carried on the spacecraft to direct sunlight [35]. This caused the battery to overheat and ultimately led to the depletion of both batteries. Incorrect antenna pointing prevented the orbiter from telling controllers its status, and its programmed safety response did not include verification that the spacecraft orientation was thermally safe [35]. While a majority of these incidents occurred due to unintended software bugs, a ransomware attack on a space vehicle, if not tactfully developed and deployed, will likely cause the same effect – cascading subsystem failures ultimately resulting in mission failure. Given the motivation of a ransomware attacker is to collect payment in receipt for the return of a functional system to the operator, it would not be in their interest to cause mission failure. Therefore, ransomware development for a space vehicle must be carefully crafted to minimize the cascading failures described; hence the research challenge.

We expect space vehicles to become increasingly susceptible to ransomware attacks in the coming decades as hardwired subsystems become obsolete. As depicted in figure 2, modern space vehicles have developed into “flying software” in the sense that nearly every subsystem now depends on software to perform its functions [36]. While this paradigm shift in space vehicle development provides significant benefit in terms of efficiency, remediating hardwired subsystems in place of software-enabled subsystems inadvertently expands the attack surface for ransomware attacks.

III. Ransomware Approach

Crafting a successful ransomware attack on a space vehicle is unique in that the victim does not possess the capability to regain physical access to the asset; whereas the attacker hijacks the space vehicle. Thus, in order for the attacker to validate the integrity of the attack and feasibility for restoration to the victim, a target must be selected which is sufficiently important, but not so critical that it disables or bricks the entire spacecraft. This is to ensure that the bare minimum systems needed for restoration are still functional. For instance, the power management system is required to ensure the spacecraft is properly conserving fuel and power. If this system is targeted, the spacecraft could hastily expend its resources and become disabled, resulting in the inability for restoration. Conversely, the target must be critical enough so that its failure compels the victim to pay the ransom.

To achieve the proposed criteria for a ransom’s impact on the space vehicle, we first performed an initial investigation of the cFS architecture through literature review of cFS publications to downselect potential targets. Next, we conducted static code analysis on the source code of the NASA cFS (version 6.6.0a) with a particular focus on downselected targets. Based on this analysis, we recorded potential cascading failures as a result of file, module, and application cross-referencing.

In ransomware attacks on conventional systems, the display is often hijacked with a locked user interface (UI) to notify the victim that their files have been encrypted and that they must pay a ransom to eradicate the malware. This hijacked display more often than not will include instructions to pay the ransom (i.e: cryptocurrency address for payment). For instance, in ransomware attacks on industrial control systems (ICS) the human-machine interface (HMI) is often exploited to serve as the display for the UI. This is not the case, however, for space-based ransomware attacks as there is no on-board display to serve as a medium for notification. While workarounds exist such as notifying the victim through third-party methods (ex: email, phone, etc.), this would only work in the context of a speared attack where this contact information is readily available. The option to omit notification altogether may be plausible in nation-state scenarios or warfare where the objective is to only cause damage, however, this is not optimal for standard cases in which the attacker seeks to have the ransom paid. Furthermore, enabling a foothold to issue a command to enable the victim to resume operations after payment can be engaged for persistence should the attacker seek to reinfect the target.

Following a ransomware attack on a spacecraft, operators are likely to initially assume that the spacecraft has malfunctioned due to unintentional causes such as software

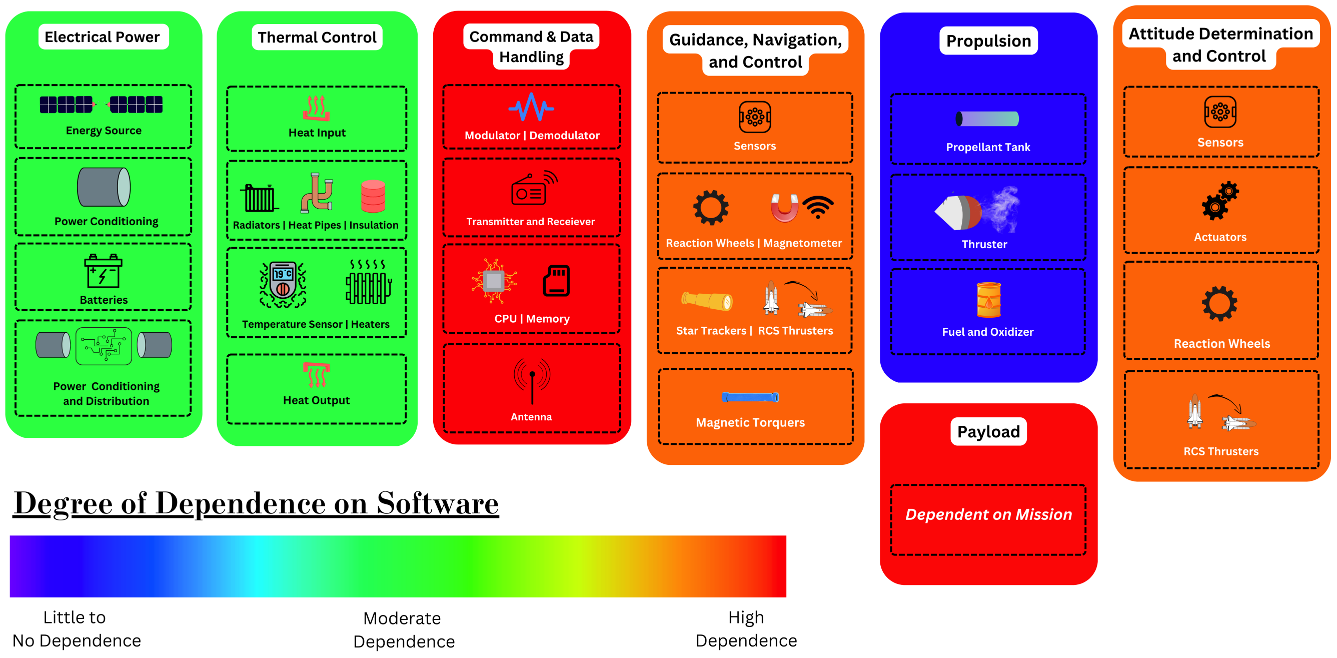

Fig. 2. **Degree of dependence on Software for key subsystems**

bugs or severe space weather. The standard incident response plan would be to subsequently retrieve a diagnostic report to examine the issue. In the NASA cFS, the application controlling this operation is the Health and Safety application (HS). While the feasibility exists to write the notification content to a value within the HS (i.e: aliveness heartbeat value), our selected target of the software bus API prevents access to the HS application following the attack. The best alternative in this case is to simply downlink the notification content prior to the ransomware deployment. Given that the attackers bear the capability of encrypting modules on the spacecraft, writing to the telemetry output application will require no additional functionality. However, in the event that the notification content is memory-intensive, there is potential risk in ensuring the swiftness of the attack. A subsequent alternative is to downlink a web address which contains the notification content. Regardless of the content of the notification, if successfully downlinked it can be assumed that operators will receive the message as it is a standard procedure to examine logs in incident response.

IV. Experiment

A. CFS Investigation

1) Overview: The NASA cFS [37] is a generic flight software architecture framework used on flagship spacecraft, human spacecraft, and CubeSats. cFS has been employed on a variety of missions such as the Lunar Reconnaissance Orbiter and the Global Precipitation Measurement Mission [38]. The software framework has also been selected for NASA's Lunar Gateway Mission and will be essential to Gateway's day-to-day operations [39].

The standard mission directory structure of cFS contains the following primary subcomponents, each of which are loaded individually by the cFS: the Core Flight Executive (cFE), Operating System Abstraction Layer (OSAL), Platform Support Package (PSP), apps, build, docs, and tools. The functionality of these subcomponents is delineated as follows.

The cFE is an application development and run-time environment which provides core services including the software bus (messaging), time management services, and table services. These core services are located within a folder titled "modules" nested within the cFE. The OSAL is a small software library that isolates the embedded software from the spacecraft's real time operating system (RTOS). High-level mission applications depend on the OSAL to interface with the spacecraft's underlying hardware. The PSP contains all the software that is needed to adapt the cFE core to a particular processor card. Each mission is expected to customize a platform support package. The apps tree contains the application source code that is shared among multiple build cPUs. Thus, this directory contains all of the high-level mission applications including the flight software for major subsystems such as the GNC. The build directory is where the cFS (cFE Core + cFS Apps) is built for a mission. It also contains all the configured mission and platform configuration files. The docs tree contains documentation including but not limited to the cFS Deployment Guide, the Telemetry and Command Mnemonic Naming Convention, and mission-specific documentation. Applications utilized to unit test cFS are stored within the tools file. All cFE APIs and OSAL APIs can be simulated with tools including but not limited to the Command Ingest (CI) Lab Tool, Telemetry Output (TO) Lab Tool, and Ground System GUI. These subcomponents compose the typical cFS mission directory structure and are by no means exhaustive. For the purpose of this investigation, we focus on the default and typical configuration of the cFS.

To downselect potential targets for static code analysis, each of the aforementioned subcomponents were assessed for damage potential. The OSAL and PSP were deemed to be too critical to afford a breach due to high potential for causing cascading failures to bare minimum systems. As the OSAL and PSP serve as the primary interfaces to the spacecraft's hardware, actuators, and sensors, a breached OSAL or PSP would likely extinguish the spacecraft's ability to access hardware or process sensor data. This would ultimately lead to a total failure of bare minimum systems. The OSAL and PSP components were consequently dismissed as potential targets. Build was also dismissed as a potential target due to empirical issues. Primarily, the build process for the cFS is typically completed prior to launch. Therefore, if the build directory was attacked, it would only take effect in instances that the spacecraft restarted in orbit and required rebuilding. Docs and tools, on the other hand, were disregarded as potential targets as their failure would not be critical enough to impose the victim to pay the ransom. This is because documentation has no direct effect on the spacecraft and tools are primarily just utilized by developers to test software prior to launch.

The remaining subcomponents (cFE and apps) were conjectured to be feasible for attack and subsequently selected for further examination. For the utility of this study, primary components within the cFE and apps directory were assessed individually. This was to ensure that any causes for cascading failures could be pinpointed. In addition, inconsequential elements such as markdown files were disregarded. Potential targets were classified into three categories: cFS application suite, mission-specific apps (GNC, AFM, SAH, etc.), and modules (cFE core services).

2) cFS Application Suite: The cFS application suite, provided by the NASA Goddard Space Flight Center, is composed of 12 individual Command and Data Handling (C&DH) flight software applications that together create a reusable library of common C&DH functions [40]. These applications serve as plug-ins to the cFE and are manually installed by developers within the apps folder of cFS. The cFS application suite includes the following C&DH applications: Checksum, File Manager, Stored Command, Limit Checker, Memory Manager, Memory Dwell, Scheduler, CFDP, Data Storage, Housekeeping, Health and Safety, and the Software Bus Network.

In conventional ransomware attacks, it is often a system's primary hard drive that is encrypted as it contains proprietary files. Victims are consequently more inclined to pay the ransom as this personal data cannot be recovered from elsewhere. Furthermore, the damage is not so critical that the system cannot power on or display the GUI. We therefore considered targeting the on-board SSD as it is the closest equivalent to a primary hard drive on a conventional computer. The enabler to the on-board SSD is the data storage application. Practically speaking, without the data storage application the cFS would be unable to write to the on-board SSD, resulting in the inability to record data onboard for downlink. After further analysis, however, it was deemed that the data storage module fell short when it came to criticality. Data recorded onboard is primarily limited to housekeeping, engineering, and science data. This data is subsequently parsed and processed to facilitate the maintenance of the spacecraft and overall mission. In the event that the availability of this application is breached, the victim can still issue commands to the spacecraft and subsequently eradicate the ransomware. Thus, a failed data storage application would likely just serve as an inconvenience and in most cases will not be significant enough to pressure the victim into paying the ransom. These factors ultimately dismissed the data storage application as an optimal target.

The file manager (FM) application was also disregarded as a potential target due to lack of criticality. Primarily, the file manager's main function is to provide a ground interface for the management of onboard files and directories. While breaching the file manager would certainly cause a hindrance to operators, concerns emerged regarding the frequency in which the file manager would be invoked. This is because the use of this application is largely dependent on the type of mission the spacecraft is intended to accomplish. To ensure the effectiveness of this attack, the FM was subsequently dismissed as a target. Based on our assessment of cFS documentation, the remaining C&DH applications within the cFS application suite were worthy of attack.

3) Mission Specific Applications: Mission-specific applications, such as the GNC, are not provided in the default configuration of the cFS. However, they are still expected to be stored within the apps directory. For the purpose of this study, we assume that fundamental mission-specific applications that enable necessary subsystems are configured. This includes applications for the following subsystems: GNC, ADCS, RF system, electrical system, and the thermal system. It is important to note that in some instances software for different subsystems may be conjoined into one application. For instance, software for the electrical system may be incorporated into the GNC or ADCS. As the plugins for fundamental mission-specific applications are not provided in cFS, attacking specific mission applications would require the malware executable to leverage signature-based detection for identification.

Completely disabling all fundamental mission-specific applications would likely result in the spacecraft to brick, due to the high criticality of certain mission applications. While the technical feasibility exists to engineer ransomware that avoids attacking critical applications, this method would primarily be of use for those that seek to conduct a speared attack on a specific victim. For the purpose of this study, we seek to provide general applicability. Thus, directly targeting mission-specific applications was dismissed as a method.

4) Modules: The modules folder within the cFE contains 14 individual subfolders. Each of these subfolders are designated to a separate module and are enumerated as follows: cfe_assert, cfe_testcase, config, core_api, core_private, executive services, event services, flight software, message, resourceid, software bus, software bus routing, table, and time. Amongst these modules, particular focus was placed on executive services and the software bus in our assessment. This was due to the elevated potential for causing cascading failures across other modules and high-level applications.

The primary subcomponent considered within the executive services module was the cfe_es_startup.scr file, which contains the startup script to the C&DH functions and subsequent mission applications. However, it was later disregarded as a target due to empirical and criticality issues. First, this script in most cases would be run prior to launch. Thus, there would be no window to attack other than whilst the spacecraft is on the launchpad (assuming the spacecraft would not be restarted in orbit). As onboard systems are actively monitored during this period, it would not be an optimal time to execute the attack as the mission would simply be postponed until the issue is resolved. As aforementioned, a major benefit for a space-based cyberattack is the victim's inability to regain physical access to their asset following deployment. For this reason, attacking the spacecraft whilst it can still be physically accessed by the victim would hinder the potency of the attack and would lessen the chances of a ransom being paid. Another issue identified with attacking the startup script is the critical cascading failures that would be caused. Namely, the GNC is directly dependent on the successful execution of this script. Thus, breaching it would disable the GNC following a restart, resulting in the inability to perform position determination. Ultimately, this led to the startup script being dismissed as a potential target.

Based on potential to cause cascading failures, the software bus (SB) was the most suited target. The SB's primary role in the cFS is to enable inter-application communication. This is accomplished utilizing a publish and subscribe functionality in which software components (publishers) produce data and egress it to the software bus. Receiving software components (subscribers), ingress this data utilizing the SB and subsequently process it. As depicted in figure 3, the software bus operates as a communication middleware (hub, intermediary, etc.) to manage the distribution of data between publishers and subscribers. There are two primary use-cases for the software bus. First, is to provide a method of ingesting uplinked commands and routing them (using the SBR module) to the appropriate location. Second, is to enable different on-board applications to interface with each other.

From a conventional software development perspective, the software bus application is therefore akin to an API, enabling different applications and features to interact with each other. Thus, it was expected that the software bus module would contain either an application or API with a primary purpose of enabling inter-process communication. From a tactical standpoint, attacking this API appeared to be very promising considering its potential for causing compartmentalization to applications relying on intermediary communication across the software bus. Thus, the API that enables the publish/subscribe functionality within the software bus was selected to be the target for static code analysis.

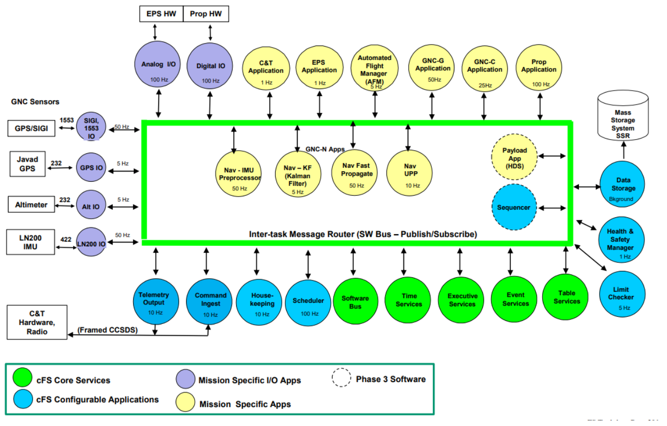

Fig. 3. Example Mission Software Architecture [33]

B. Static Code Dependencies for Modules

Static code analysis is a method of white box testing in which source code is examined without execution. In addition to serving as a technique for debugging and addressing logic problems, static code analysis is commonly utilized to inspect and verify the inner workings of a software system and integrations with external systems.

The NASA cFS is designed to be highly modular and customizable to satisfy a wide range of use-cases. In mission-specific instances, it is therefore common that sets of execution paths, files, variables, and/or modules are unreferenced during execution. Conducting static code analysis as opposed to dynamic code analysis provides benefit in this case, as it ensures that our analysis is all-encompassing. Specifically, file paths or data files that are unreferenced during execution or not being actively accessed in mission-payloads can still be assessed for target selection.

A myriad of static code analysis software tools exist across the market, each with unique libraries optimized for performing specific types of examinations. While these tools are primarily employed for resolving issues related to lack of adherence to project regulations/protocols, a subordinate benefit is the ability to enhance the user's understanding of the code structure. In identifying cascading failures, selecting a tool which excels at this service provides great value. Consequently, static code analysis software tools were assessed based on the metric of ability to increase insight into the architecture of the code. Tools that were considered included SonarQube, PMD, Understand 2.0, and CodePeer. Ultimately, Understand 2.0 proved to the best fitted for our use-case.

To initiate our analysis, a diagram was generated to architect the code structure of the software bus module. Nested within the flight software folder and source folder was an API titled "cfe_sb_api.c". Other identified files included header files for the software bus module in addition to buffer, task,

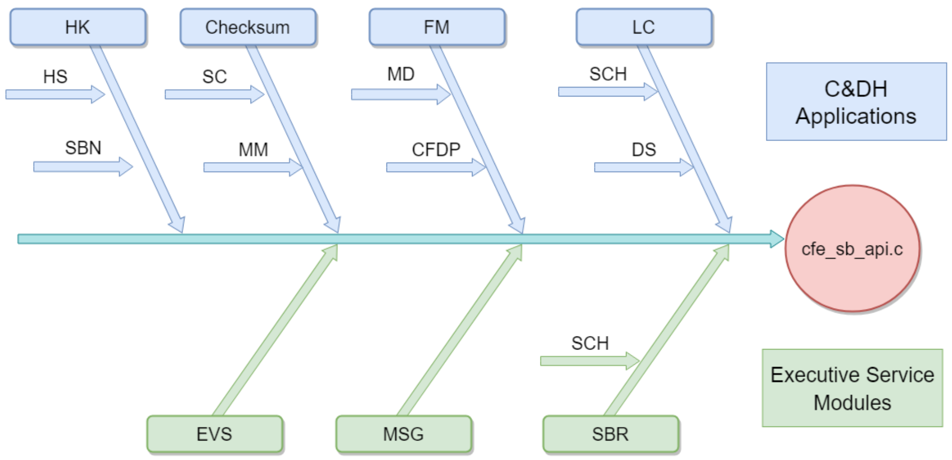

Fig. 4. **Cascading Failures from Attacking the Software Bus API**

and initialization files. The cfe_sb_api.c (software bus API) contained the source code which enables the publish and subscribe functionality.

To obtain a more thorough insight into cascading failures, two portions of static code analysis commenced. First, we inspected the underlying functionality of the API to examine how it enabled different applications to interface via the software bus. Subsequent analysis was conducted to identify what would occur to reliant applications if this functionality was disabled from our attack. Next, we formulated a diagram utilizing Understand 2.0 to identify all the specific calls to the software bus API across cFE core modules and the cFE application suite (C&DH components). The outputted data was utilized to record all the probable cascading failures that would occur from attacking the software bus API.

There are multiple functions within the software bus API that enable inter-application communication. This process is first initiated by calling the software bus API's CFE_SB_CreatePipe function, in which an application notifies the cFE of pipes that it seeks to receive data. Most mission-applications and C&DH applications will first call this function whilst initializing the application. Failure to effectively create a pipe (initiate inter-app communication) will return an error. C&DH applications were found to return this error directly to the event services module whereas mission-specific applications returned to one of the four available/enabled cFE ports (os_printf). There are some empirical issues with attacking just the CFE_SB_CreatePipe function as opposed to the whole API. While this function must be called to initiate communication between apps, it is usually called once (immediately succeeding startup and configuration of the application). Attacking this target whilst the spacecraft is in orbit may consequently not have desired effects, as the function is not usually invoked continuously. Conversely, if the spacecraft is to restart or a new application is uplinked following the attack, the ransomware will function as intended.

Succeeding the creation of a pipe, an application must indicate to the software bus that it seeks to ingress data into the respective pipe. This is referred to as a ”software bus message subscription” and requires the calling of the following function: CFE_SB_Subscribe. In contrast to CFE_SB_CreatePipe, this function is often continually engaged so that applications can receive real-time data. This is especially true of applications that require updated sensor data to enable operations. All applications within the cFS application suite and modules folder called this function within their source code. While the use-case varied across these applications, many called the subscription function to subscribe to housekeeping messages and ground commands. Without access to subscription functionality, IPC will be disabled for all applications stored within the apps directory.

To initiate the transmission of requested data over the software bus to subscribed applications, an application will perform the publishing operation by calling the CFE_SB_TransmitMsg function. Respective data is then sent to the software bus messaging queue to be available for those that have subscribed. Based on the call structure, if this function is disabled no application will be able to publish data to the software bus. As a result, the software bus messaging queue will eventually clear following the attack and there will be no available data to subscribe to. Thus, data transfer over the software bus will be disabled and will result in a communication void between applications. Note that this cascading failure will only cause enforced compartmentalization to applications stored within the cFS/apps directory. Components that do not require the software bus such as those communicating with serial are expected to function nominally.

The final step is the data ingress to an application using the CFE_SB_ReceiveBuffer function. Akin to the subscription function, ReceiveBuffer was also found to be utilized ubiquitously to assist in processing data that has been requested. Since most applications are message-driven, this primarily occurred in the application's main execution loop and was consequently engaged continuously. Due to call order, this function will remain futile in the event that the aforementioned functions have been disabled as they are the primary mechanisms to initiate the ingress of data from the software bus.

Based on our analysis, preventing access to the software bus API is bound to cause cascading failures to all applications stored within cFS/apps that require inter-app functionality. While potential workarounds exist to prevent abrupt failure to applications (such as implementing error handling), compartmentalization will be forced regardless, significantly hindering mission accomplishment.

To formulate a call diagram for the software bus API, we first configured cFS to include all default modules and the cFS application suite (C&DH applications). Understand 2.0 was subsequently utilized to output all cross references between default modules, the cFS application suite, and the software bus API. As depicted in figure 4, all C&DH applications were found to reference the software bus API. Moderate cascading failures were also identified for components in the executive services modules. This specifically includes software bus routing, event services, scheduler, and message.

V. Results

Based on our analysis, the software bus API (cfe_sb_api.c) was found to be the most optimal target for a ransomware attack given its moderate damage potential and feasibility for attack. Due to the extensive utility of this API for C&DH functionality and mission-specific applications, it is likely that holding this target hostage would significantly hinder mission-accomplishment and ultimately impose the victim to pay the ransom, while still enabling the recovery of the space vehicle.

VI. Discussion

There are several limitations of this study, which are important considerations for future work. First, our analysis was completed on cFS which is not ubiquitous, despite being a popular, open-source flight software framework. Therefore this approach may not be scalable to other flight software. Second, we were unable to account for all cascading failures due to the variability/customizable nature of cFS and spacecrafts in general. For example, some subsystems may communicate via serial interface and will not be affected by certain attacks, but in other instances critical subsystems may be configured to rely on the software bus for inter-process communication (IPC) and will therefore cause cascading failures to those operations. Finally, the static code analysis does not empirically verify findings; however is intended as a first step towards outlining an empirical experiment.

Future research includes conducting dynamic code analysis to verify the results of this study. Further, our team intends to develop a malware kit to demonstrate the feasibility of a ransom attack against a space vehicle using the outlined approach. This kit will feature a malicious cFS application which is tasked with holding the software bus API hostage by flooding it with messages. The CFE_SB_TransmitMsg function will be excessively engaged and exploited to enable this operation. The following code sample provides a basic overview of the malware's functionality.

```
{Application entry point}
void CustomAppMain(void) {
  CFE_MSG_Message_t customMsg;
  CFE_MSG_Init(&customMsg, CUSTOM_MSG_ID,
sizeof(CustomMsg_t));
{Main application loop}
  while (1) {
    for (int i = 0; i < 100000; i++) {
      CFE_SB_TransmitMsg(&customMsg, true);
}
```

The malware leverages an infinite loop to send 1,000,000 custom messages across the software bus every second. This will fully occupy the software bus API and prevent other applications from sending or receiving messages. While this code sample does not feature the destination of the message, as it would generally be designated in the customMsg structure, the destination is intended to be the housekeeping pipe to maximize the attack surface.

As part of this exercise, we must evaluate the mechanics for remediating the malware's impact following the ransom payment. Next steps also include standardizing the attack vectors described in the ransom approach using consistent terminology outlined in the Space Attack Research and Tactic Analysis (SPARTA) framework [41].

VII. Conclusion

This study establishes an approach for hijacking and ransoming a space vehicle as discovered through conducting a static code analysis on open-source flight software. Through a dependency analysis, we assessed that it is viable to launch a ransom attack on a space vehicle without creating a space brick in the process, enabling the satellite's recovery after a ransom is paid.

References

[1] G. Falco, "Cybersecurity principles for space systems," *Journal of Aerospace Information Systems*, vol. 16, no. 2, pp. 61–70, 2019.

[2] G. Falco, W. Henry, M. Aliberti, B. Bailey, M. Bailly, S. Bonnart, N. Boschetti, M. Bottarelli, A. Byerly, J. Brule *et al.*, "An international technical standard for commercial space system cybersecurity-a call to action," in *ASCEND 2022*, 2022, p. 4302.

[3] G. Falco, A. Noriega, and L. Susskind, "Cyber negotiation: A cyber risk management approach to defend urban critical infrastructure from cyberattacks," *Journal of Cyber Policy*, vol. 4, no. 1, pp. 90–116, 2019.

[4] G. J. Falco and E. Rosenbach, *Confronting Cyber Risk: An Embedded Endurance Strategy for Cybersecurity*. Oxford University Press, 2022.

[5] E. Newburger, "Ransomware attack forces shutdown of largest fuel pipeline in the U.S." May 2021. [Online]. Available: https://www.cnbc.com/2021/05/08/colonial-pipeline-shuts-pipeline-operations-after-cyberattack.html

[6] N. Perlroth, "Colonial Pipeline paid 75 Bitcoin, or roughly $5 million, to hackers." *The New York Times*, May 2021. [Online]. Available: https://www.nytimes.com/2021/05/13/technology/colonial-pipeline-ransom.html

[7] "US fuel pipeline hackers 'didn't mean to create problems'," *BBC News*, May 2021. [Online]. Available: https://www.bbc.com/news/business-57050690

[8] J. Bunge, "WSJ News Exclusive | JBS Paid $11 Million to Resolve Ransomware Attack," *Wall Street Journal*, Jun. 2021. [Online]. Available: https://www.wsj.com/articles/jbs-paid-11-million-to-resolve-ransomware-attack-11623280781

[9] "Cyberattack hits world's largest meat processor's Australia and North America units," Jun. 2021. [Online]. Available: https://www.cnbc.com/2021/06/01/cyber-attack-hits-jbs-in-australia-and-north-america.html

[10] J. Creswell, N. Perlroth, and N. Scheiber, "Ransomware Disrupts Meat Plants in Latest Attack on Critical U.S. Business," *The New York Times*, Jun. 2021. [Online]. Available: https://www.nytimes.com/2021/06/01/business/meat-plant-cyberattack-jbs.html

[11] A. B. C. News, "JBS paid $11 million in ransom after cyberattack, company says." [Online]. Available: https://abcnews.go.com/Business/jbs-paid-11-million-ransom-cyber-attack-company/story?id=78185017

[12] T. Johnson and M. W. Bureau, "Here's one tally of the losses from WannaCry cyberattack." [Online]. Available: https://phys.org/news/2017-05-tally-losses-wannacry-cyberattack.html

[13] "5 Biggest Ransomware Attacks in History," Sep. 2021. [Online]. Available: https://swisscyberinstitute.com/blog/5-biggest-ransomware-attacks-in-history/

[14] S. Ghafur, S. Kristensen, K. Honeyford, G. Martin, A. Darzi, and P. Aylin, "A retrospective impact analysis of the WannaCry cyberattack on the NHS," *npj Digital Medicine*, vol. 2, no. 1, pp. 1–7, Oct. 2019. [Online]. Available: https://www.nature.com/articles/s41746-019-0161-6

[15] G. Cohen, "Throwback Attack: WannaCry ransomware takes Renault-Nissan plants offline," Apr. 2021. [Online]. Available: https://www.industrialcybersecuritypulse.com/facilities/throwback-attack-wannacry-ransomware-takes-renault-nissan-plants-offline/

[16] "NotPetya: the cyberattack that shook the world." [Online]. Available: https://economictimes.indiatimes.com/tech/newsletters/ettech-unwrapped/notpetya-the-cyberattack-that-shook-the-world/articleshow/89997076.cms

[17] A. Greenberg, "The Untold Story of NotPetya, the Most Devastating Cyberattack in History," *Wired*. [Online]. Available: https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/

[18] D. G. July 12 and 2021, "CNA Discloses Breach Related to March Ransomware Attack." [Online]. Available: https://www.bankinfosecurity.com/cna-discloses-breach-related-to-march-ransomware-attack-a-17022

[19] "Ransomware gang breached CNA's network via fake browser update." [Online]. Available: https://www.bleepingcomputer.com/news/security/ransomware-gang-breached-cna-s-network-via-fake-browser-update/

[20] K. Mehrotra and W. Turton, "CNA Financial Paid $40 Million in Ransom After March Cyberattack," May 2021. [Online]. Available: https://www.bloomberg.com/news/articles/2021-05-20/cna-financial-paid-40-million-in-ransom-after-march-cyberattack

[21] "CNA Financial Pays $40 Million Over March Ransomware Attack, the Highest Known Ransom Payout." [Online]. Available: https://www.spiceworks.com/it-security/vulnerability-management/news/cna-financial-pays-40-million-over-march-ransomware-attack-the-highest-known-ransom-payout/

[22] P. Arntz, "Ransomware revenue significantly down over 2022," Jan. 2023. [Online]. Available: https://www.malwarebytes.com/blog/news/2023/01/ransomware-revenue-significantly-down-over-2022

[23] S. C. Staff, "Ransomware deployable in embedded systems," Jan. 2022. [Online]. Available: https://www.scmagazine.com/brief/device-security/ransomware-deployable-in-embedded-systems

[24] D. Storm, "Hackers demonstrated first ransomware for IoT thermostats at DEF CON," Aug. 2016. [Online]. Available: https://www.computerworld.com/article/3105001/hackers-demonstrated-first-ransomware-for-iot-thermostats-at-def-con.html

[25] C. Brierley, J. Pont, B. Arief, D. J. Barnes, and J. Hernandez-Castro, "PaperW8: an IoT bricking ransomware proof of concept," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*. Virtual Event Ireland: ACM, Aug. 2020, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3407023.3407044

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

[27] B. N, "Redirecting — cybersecuritynews-com.cdn.ampproject.org," https://cybersecuritynews-com.cdn.ampproject.org/c/s/cybersecuritynews.com/ransomware-groups-attacking-satellite/?amp, [Accessed 01-Jun-2023].

[28] "Operation Cataclysm - Team OneFist And MegaFon Satellite Network," Oct. 2022. [Online]. Available: https://www.cyberthreat.report/operation-cataclysm/

[29] D. R. S. ReadingMarch 14 and 2023, "LockBit Threatens to Leak Stolen SpaceX Schematics," Mar. 2023. [Online]. Available: https://www.darkreading.com/ics-ot/lockbit-threatens-leak-stolen-spacex-schematics

[30] "Hackers use Conti's leaked ransomware to attack Russian companies." [Online]. Available: https://www.bleepingcomputer.com/news/security/hackers-use-contis-leaked-ransomware-to-attack-russian-companies/

[31] E. Browne, "Roscosmos Head Rejects Anonymous Claim That Russian Satellites Were Hacked," Mar. 2022. [Online]. Available: https://www.newsweek.com/roscosmos-head-dmitry-rogozin-anonymous-russian-satellite-hack-1684033

[32] "https://twitter.com/xxnb65." [Online]. Available: https://twitter.com/xxnb65

[33] "Core Flight System." [Online]. Available: https://cfs.gsfc.nasa.gov/

[34] P. J. ITworld, Writer/Editor at, "8 Famous Software Bugs in Space," Feb. 2013. [Online]. Available: https://www.csoonline.com/article/3404528/8-famous-software-bugs-in-space.html

[35] N. Administrator, "Report Reveals Likely Causes of Mars Spacecraft Loss," Jun. 2013. [Online]. Available: http://www.nasa.gov/mission_pages/mgs/mgs-20070413.html

[36] *Understanding Space An Introduction to Astronautics and Space Systems*. Teaching Science and Technology. INC.

[37] Nasa, "Nasa/cfs: The core flight system (cfs)." [Online]. Available: https://github.com/nasa/cFS

[38] "Core Flight System — cfs.gsfc.nasa.gov," https://cfs.gsfc.nasa.gov/Introduction.html, [Accessed 01-Jun-2023].

[39] "Core Flight Software Chosen for Lunar Gateway—nasa.gov," https://www.nasa.gov/feature/goddard/2021/core-flight-software-chosen-for-lunar-gateway, [Accessed 01-Jun-2023].

[40] R. Garner, "NASA Goddard Releases Open Source Core Flight Software System Applicat," Mar. 2015. [Online]. Available: http://www.nasa.gov/press/goddard/2015/march/nasa-goddard-releases-open-source-core-flight-software-system-application-suite-to

[41] A. Corporation, "Space attack research and tactic analysis (sparta)." [Online]. Available: https://sparta.aerospace.org/