Senior Design Final Report

Satellite Anomaly Injection & Detection Testbed (SAID Testbed)

Version 1.0 - 05/11/2021

Team Members:
Matthew Gilligan
Alex Huang
Alexander Lopez
Jerome Pineda
Vivian Sau
Samantha Simpson
Aaron Tong
Nicholas Torres
Joshua Tran

Faculty Advisor:
Russ Abbott

Liaisons:
Rick Johnson
Denny Ly
Karina Martinez
Pablo Settecase

Table of Contents

1. Introduction:

1.1. Purpose

The Aerospace Corporation is a nonprofit organization that provides technical guidance and advice on all aspects of space missions to military, civil and commercial customers. Aerospace's specialization is in satellites. Satellites are used for communications, weather, geolocation, defense, and many others. It is crucial to monitor the satellite's health and ensure accurate data are being sent to and from the satellite. When anomalies occur, it is vital to detect and fix the error before a disaster happens. Consequences can include losing communication, downlinking the wrong information, or complete mission failure leading to a significant expense.

The Satellite Anomaly Injection and Detection Testbed, abbreviated as the SAID Testbed, is a project that uses OpenSatKit (OSK) - a complete desktop solution for learning how to use NASA's core Flight System. OSK combines NASA's core Flight System, Ball Aerospace Corporation's COSMOS ground system, and NASA Goddard's 42 satellite simulator. OSK simulates the life of a satellite and, more importantly, acts as the communication between satellite and ground system

Our goal for SAID is to inject and detect five types of anomalies using the OSK environment. The abnormalities include Denial of Service, Invalid Command Sequences, Memory Leak, Runaway Task, and Single Bit Error.

- **Denial of service:** occurs when network communication services are interrupted or shut down, making them inaccessible to the intended users. Denial of service often happens when a network floods with traffic. Our solution to a denial of service is to develop a program that continually monitors the network traffic. Once an increase in traffic is detected, it will send an alert to the ground system.
- **Invalid Command Sequence:** is a series of commands that are not relayed in the correct order. Invalid command sequence can be a result due to human error or human interference. A simple example of an invalid command sequence is a satellite that has a camera on it. A correct series of commands to take a picture with this satellite would be to 1. Power on camera, 2. Take a picture, 3. Power off camera. An invalid command sequence would shuffle these commands in the incorrect order, such as 1. Take a picture, 2. Power on camera, 3. Power off the camera. The solution to detection is to monitor the sequence of commands before they are sent up to the satellite.

- **Memory leak:** is when the memory on the satellite is not released after use as it usually should but continues to occupy resources uncontrollably and slowing overall operations. Our solution is to monitor the memory usage of the satellite. If the usage is above average, a memory leak may be occurring.
- **Runaway Task:** is when a task fails to terminate, causing it to use up the CPU on board and not have enough resources to carry out other essential tasks. Our solution is to monitor the CPU performance of the satellite. If the usage is above expected, a runaway task may be occurring.
- **Single Bit Error:** is when the onboard memory of the satellite experiences an unexpected change in its data. Single Bit Error happens due to the space weather environment. Our solution to this anomaly is to monitor the state of the data in memory continually.

1.2. Design Principles

The anomaly software is the main deliverable of this project while utilizing the OSK environment. Our main principle is to utilize OSK to its fullest using its API to smoothen the design experience. Although this was our intention, it was challenging to use OSK and build around in the actual process. Therefore, another principle came out: keeping in mind the individuals not familiar with the environment. The need to provide an efficient User Manual is crucial. The manual must showcase OSK in a simple, intuitive manner. We also need to keep in mind that future maintenance and expansions will not be too complicated. Our main goals are to create demos and user manuals to showcase how to utilize the software components.

1.3. Design Benefits

The benefit of utilizing OSK is that we did not need to create new code and instead understand how their code works. By doing that, it streamlines the process of creating the applications that we want. However, the difficulty that arose from this situation is the lack of documentation on OSK. There were no design documents that specified how to create new applications on OSK and how it can utilize its API. This is why our second principle of keeping in mind individuals not familiar with the environment came about. With the documentation that we have, we can smoothen out the experience of future developers with a lot less scrounging around to find a way to code and instead instantly allowing them to code and get their feet wet.

1.4. Achievements

Over the academic year, the team was able to document how to develop applications in the OSK environment. The leading issue with OSK is that there is limited documentation on how to develop applications on it. One of our team members connected with the developers of OSK and documented any notes that can help streamline the development process. Furthermore, one of our team members made a step-by-step guide on how to develop commands on OSK.

The team was able to provide three out of the five anomaly injections. The anomalies in question are the Denial of Service (DOS), Single Bit Error, and Runaway Task. The other two anomalies, Memory Leak, and Invalid Command Sequences, have been extensively researched and have documented the conceptual routes that can be taken for the next team to handle this project.

The team was able to provide two out of the five anomaly detections. The anomaly detections in question are the Denial of Service (DOS), and Runaway Task. The other three anomalies, Memory Leak, Invalid Command Sequences, and Single Bit Error, have researched ways of approaches in order to detect their anomalies. This is done so the next team that decides to tackle this project has some kind of incite of how to handle the project.

2. Related Technologies:

2.1. Existing Solutions:

Our team looked into different avenues on how to approach anomaly detection. One of the leading avenues of detection is machine learning. Machine learning allowed a program to learn from big data and start picking out outliers. This was the case for the PyOD software we have found. PyOD is a python oriented program able to detect outliers within data.

However, the limitation to machine learning is how much does the team knows about it. Unfortunately for our team, we had limited knowledge on how machine learning. Since we still needed to learn the ins and outs of the OSK environment, which was proving complicated, we decided to scrap the idea of using machine learning.

The next avenue that we took a look into is OSK itself. OSK has a variety of ways to check if limits have been reached. This includes disconnection of the program if the network has been flooded, a checksum app that allows the check of the state of the memory, and other ways to see the data we need.

With the limited knowledge of machine learning, we opted to use the already existing limiters found in OSK. We were also required to build our limiters if the function were not built into OSK yet.

2.2. Reused Products:

Both anomaly injection and detection are developed on C using the framework provided by OSK. We used the API found on OSK and configured it to allow us to inject and detect anomalies.

3. System architecture:

3.1. Overview:

The architecture for the Anomaly Injection, Ground Base Anomaly Detection (GBAD), and the Onboard Anomaly Detection (OAD) is situated on where the program would make sense in the simulation. The Anomaly Injection and Onboard Anomaly Detection (OAD) is hooked up on the Core Flight System (cFS), where the simulated satellite is located. The Ground Base Anomaly Detection (GBAD) is situated on COSMOS, where the simulated ground system is.

Here is a diagram (DFD level 1) that shows the architecture.

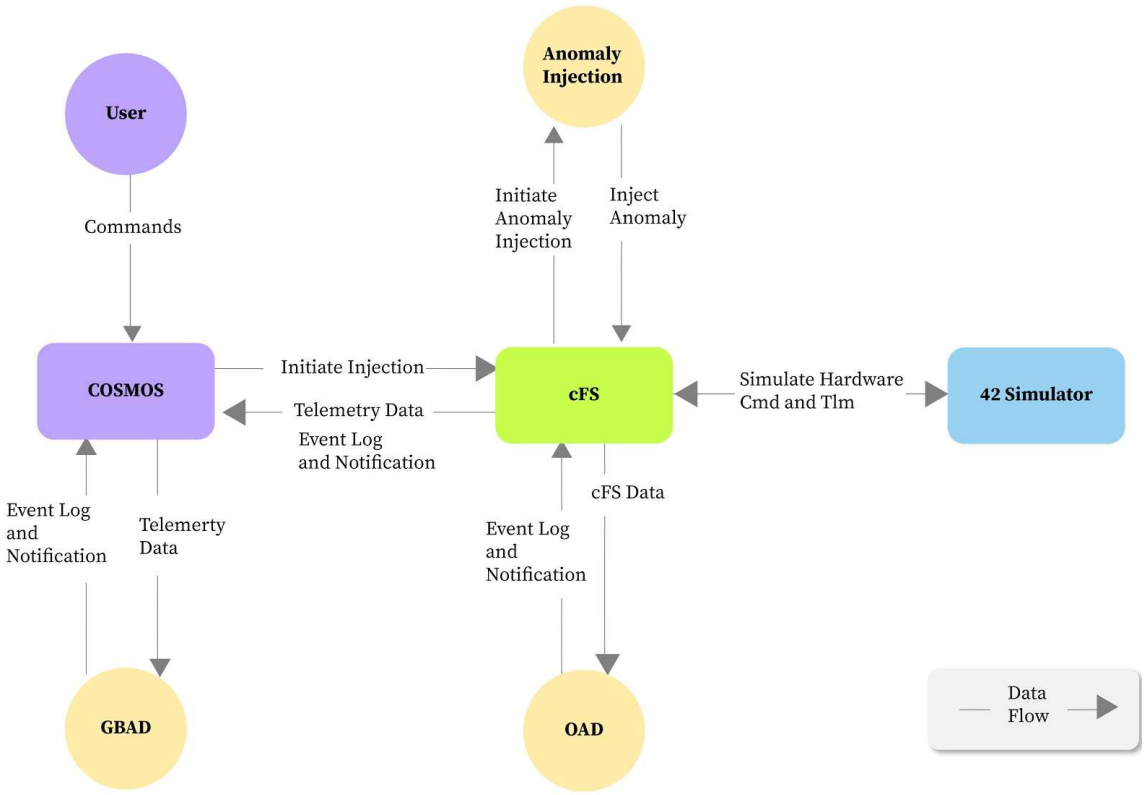

Figure 3 – 1 Level 1 Data Flow Diagram

- **COSMOS:** COSMOS is the built in program of OSK that acts as the ground system. The user is able to communicate to the satellite only through COSMOS. Any commands that is sent by the user will go into COSMOS and will be sent up into the cFS where it will be processed and go to its respective application. COSMOS is also in charge of downlinking and kind of telemetry data that is to be downlinked

- **Core Flight System (cFS):** cFS is a built in program of OSK and acts as the flight software framework. This is where we can develop any kind of applications that can communicate to the ground and satellite. It also has pre built applications that handles the satellites insides such as the memory, checking its health and safety, and allowing the transfer of files from satellite to ground.
- **42 Simulator:** 42 is the built in program of OSK and acts as the graphical simulator of the satellite. In 42 it shows the satellite orbiting around the Earth and states the altitude of the satellite. It also has a graphic of the map and the trajectory of where the satellite is going.
- **Anomaly Injection**: This is where we ended putting all the anomalies that we want to activate. Whenever we want to trigger an anomaly a command coming from COSMOS is sent to the cFS and then transferred to the choice anomaly. Then the anomaly infects the relative system in the cFS.
- **Onboard Anomaly Detection (OAD)**: The OAD is activated on the launch of the simulation. It can do background checks of anomalies by being able to gather data from the cFS. Whenever an abnormality is detected, an event log and notification is sent back to COSMOS, notifying the user.
- **Ground Base Anomaly Detection (GBAD)**: The GBAD is activated on launch of the simulation. It is able to do background checks of anomalies by gathering data from the telemetry. Whenever an anomaly is detected an event log and notification is created and is sent back to COSMOS notifying the user.

3.2. Implementation:

To implement each anomaly and its detection, we split the group into five, each correlating with the anomaly given. Each anomaly is then subdivided into two sub-sections, one on injection and the other is on detection.

3.2.1 Anomaly Injection:

From the five anomalies given three were injected successfully those were the Denial of Service, Single Bit Error, and Runaway Task.

3.2.1.1. Denial of Service (DOS): DOS made use of a toolbox called Netwox. Netwox allows the spamming of packets containing data units. This spamming is direceted towards the network of OSK.

3.2.1.2: Single Bit Error: Single Bit Error made use of the core Flight Executive (cFE) API and the XOR operator. The process was to use the cFE API to grab the memory of the satellite, use an XOR operator to flip one bit of that memory, and then plug in that flipped memory back into the satellite.

3.2.1.3: Runaway Task: Runaway Task made use of the making of orphan threads. This was done by forking a thread and not letting it come back to the main thread.

3.2.2 Anomaly Detection:

From the five anomalies given two were detected successfully, those were the Denial of Service, and Runaway Task.

3.2.2.1 Denial of Service (DOS): Keeps monitor of the network traffic. It does this by checking the a file called rx bytes file. This file indicates the volume of data packets received in bytes. The network transmission speed is measured in KiloBytes Per Second (kpbs). If the transmission speed exceeds a certain rate the app will send a message to notify the user of the unusual amount of traffic

3.2.2.2 Runaway Task: Keeps monitor of the of resource consumption of the satellite.

4. Conclusions:

4.1. Results:

Our team developed three out of the five anomalies that Aerospace inquired about under the OSK environment. The Denial of Service, Single Bit Error, and Runaway Task. The Denial of Service and Runaway task achieve both injection and detection purposes. The Single Bit Error accomplishes injection and provides details for the detection. As an added note, all applications are built on the simulated satellite, cFS.

4.2. Future:

As it can be seen by the results there is still a lot of worked that is needed to be done. If this project is to be developed in the future the rest of the anomalies are still needed to be implemented and this especially applies to the detection of said anomalies. As discussed in section 2.1 there are existing technology that can do anomaly detection and that is machine learning. If a team were to pick this project up the following is recommended to be done

- If it is possible learn about machine learning software, use it for the anomaly detection. As of right now we planned on using OSK as a way to do the detections. This proved to be harder than expected and we suspect using a machine learning program, such as PyOD, would be able to streamline the process of detection. This will also add the benefit of adding a more general detection of anomalies and not just the ones we built
- As of right now the anomaly injections and anomaly detections are not bundled up into one application. Due to the OSK environment we needed to separate them. It might be possible to bundle all of the injections into one application and bundle all the detection into another application.
- Adding a feature that allows the executions of anomalies at random. This could be accomplished using the House Keeping app to wake up the anomalies at random. This will also help on checking if detection works.
- Create a distinction between ground and onboard detection/injections. As of right now all of our systems are built on the cFS which is the simulated satellite. The reasoning behind this is because it was the simplest way to actually create the applications. If it is possible being able to create the distinction between onboard and on ground anomalies injection and detection would create a more accurate simulation.

5. References:

Core Flight System (cFS): https://cfs.gsfc.nasa.gov/

COSMOS: https://cosmosc2.com/

Open Sat Kit (OSK): https://github.com/OpenSatKit/OpenSatKit/wiki