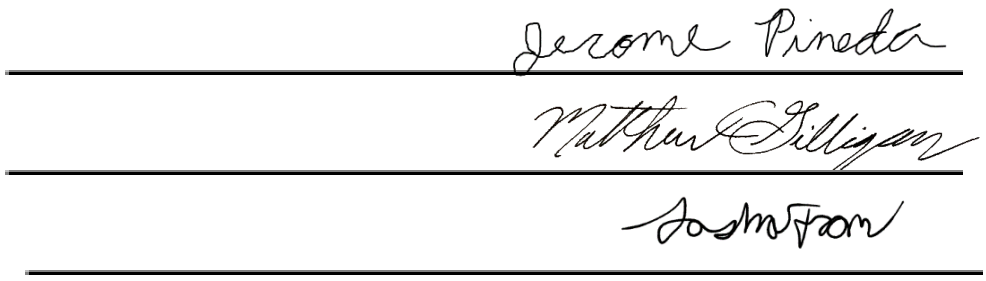
Software Design Document

for

Satellite Anomaly Injection & Detection Testbed

Version 1.0 Approved



Prepared by Alex Huang, Jerome Pineda, Samantha Simpson, Nicholas Torres, Joshua Tran

CSULA / The Aerospace Corporation

May 14, 2021

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Architecture & Requirements Committee	5/14/2021	Initial Draft	Version 1.0

1. Introduction

1.1. Purpose

This document serves to record how the requirements for the SAID Testbed shall be implemented. It will expand on the Software Requirements Specifications' functionalities and provide details on how each module is to be executed, their purpose, and how they all interact with each other. This document's completed state will be ready for the development team to develop the expected software.

1.2. Document Conventions

Bold lettering signifies a new section and shall have relevant information below. Font size shall indicate the hierarchical structure of content with larger fonts indicating general concepts and smaller fonts indicating more specific content.

1.3. Intended Audience and Reading Suggestions

This Software Requirements document is intentional for:

- Software Developers can review the project's capabilities and more easily understand where their efforts should target to improve or add more features to it.
- Project testers can use this document as a base for their testing strategy as some bugs are easier to find using a requirements document. This way of testing becomes more methodically organized.
- Project reviewers can use this document as a base for analyzing and critiquing design plans.
- Project managers can use this document to help the project team plan how they should implement its quality control.
- End-users of this application who wish to read about what this project can do.
- End-users can use this document to read the capabilities of the application.

2. Design Considerations

2.1 Assumptions and Dependencies

- Ubuntu Operating System (18.04 LTS)

2.2 General Constraints

- Onboard Anomaly Detection (OAD). CPU on the satellite has limited cycles and a processor that runs on a single thread. The OAD needs to be lightweight and not consume too much power.

2.3 Goals and Guidelines

- Use as much open source software as possible to reduce engineering time.
- Employ KISS (“Keep it simple stupid”) principle as much as possible.
- Research and understand possible system anomalies.
- Design with modularity in mind for ease of rapid development.
- Design injections purely for demonstration purposes.
- The software will aim to run detection capabilities on its own.
- Goal is to complete the project by the end of the Spring 2021 semester.

2.4 Development Methods

A combination of Agile and Waterfall development methods.

- Waterfall methodology as we gather information from Aerospace and become more acquainted with the necessary software. As well as research to develop a clear understanding of anomalies and normalcy of desired features. Sequentially the project plan is created to accommodate those requirements.
- Agile software development methods as we work collaboratively between committees. Requirements and solutions will evolve through collaboration between self-organizing cross-functional teams.

3. Software Overview

3.1. Software Components

Reference SRS document section *3.3 Software Interfaces*.

3.2. Software Deliverables

Anomaly Injection, Onboard Anomaly Detection, and Ground Based Anomaly Detection are the three software deliverables the SAID Testbed team is developing.

3.2.1. Anomaly Injection

The anomaly injection unit allows the user to inject anomalies into the OSK environment. Anomaly injection has an automated process that will inject a random anomaly at a random time. The user will have control to turn on and off the automated process.

3.2.2. Onboard Anomaly Detection

The onboard anomaly detection system will contain lightweight routines (in the form of an OSK app) to handle the detection of single-bit errors, runaway tasks, denial of service, and memory leaks. Each error shall contain its method of detection separate from one another.

3.2.3. Ground Based Anomaly Detection

The ground-based anomaly detection shall contain a process to detect invalid command sequences entered by the user. Invalid command sequences are a string of commands that would cause unintended behavior from the satellite to occur. An example of this would be a command to rotate a satellite arm in a direction that is impossible for the satellite. The detection system will detect and notify the user of such sequences.

4. Softwares Architecture

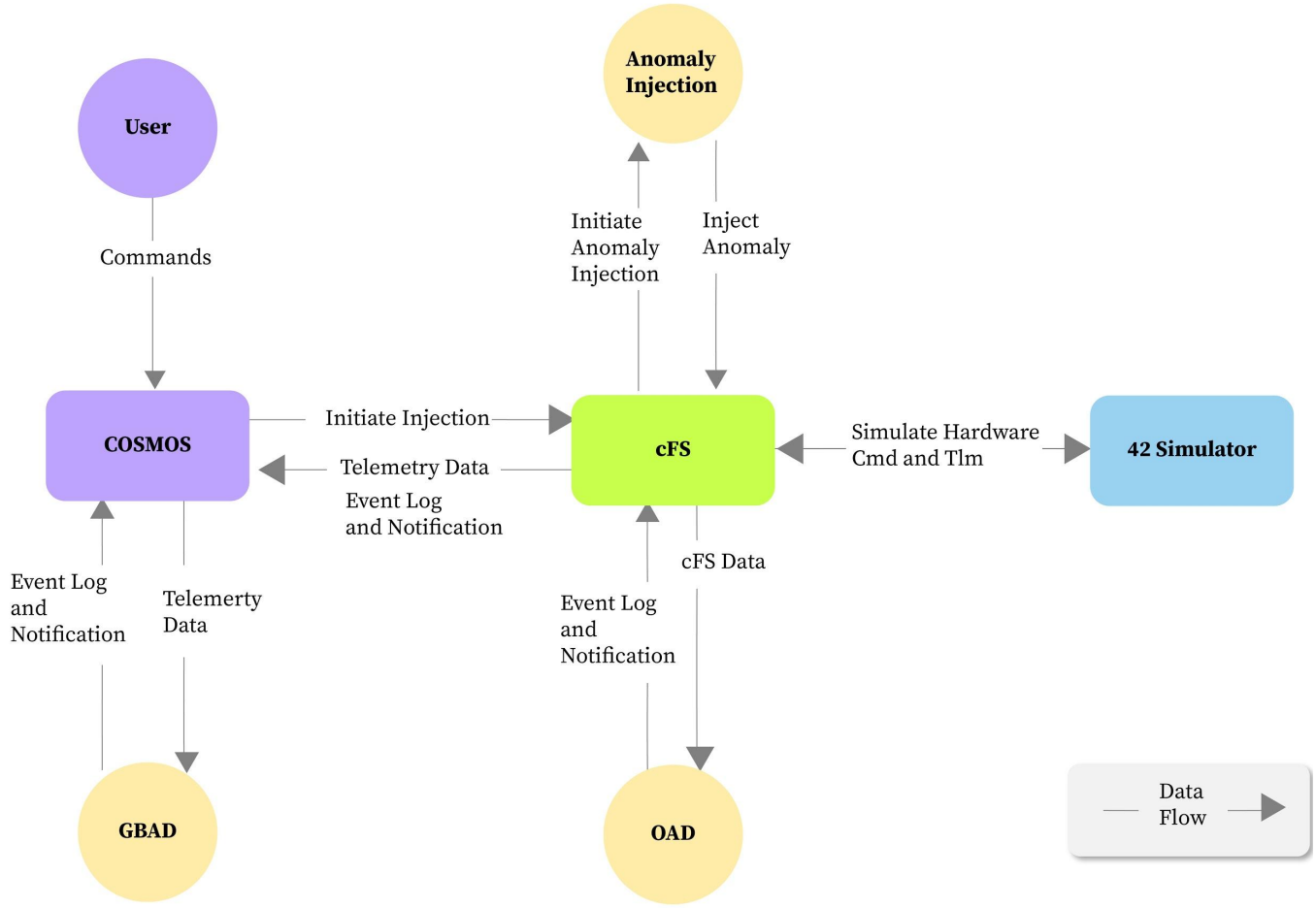


Figure 4 - 1 Level 0 Data Flow Diagram

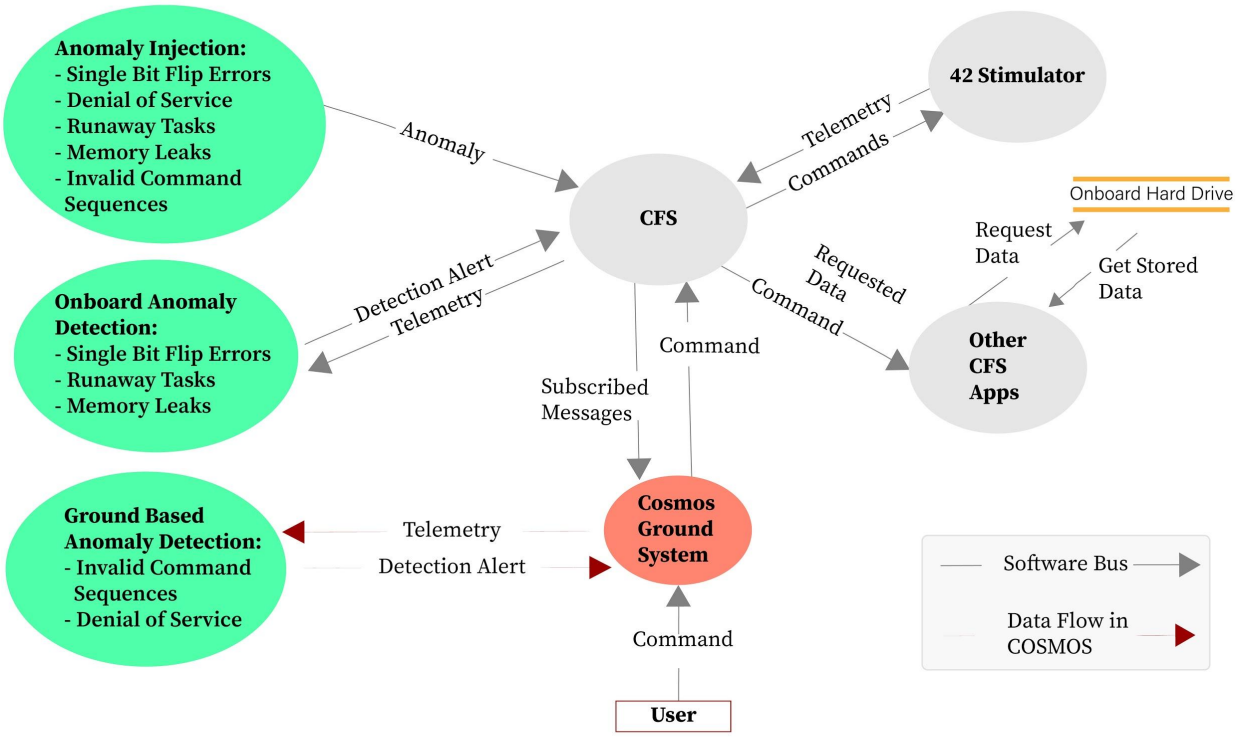


Figure 4 - 2 Level 1 Data Flow Diagram

5. Detailed System Design

5.1. Anomaly Injection

The anomaly injection software shall allow the user to trigger a one-off anomaly of their choice by entering a command line or the command and telemetry server of OSK. This set of commands will be made using the custom command reference on the OSK GitHub repository. The injection will then be executed in the injection module of the deliverables for the chosen command. The API that shall be used for anomaly injection shall include the software bus for inter-application communication and anomaly injection.

Note: Any anomaly that is created will affect the OSK system. OSK has a built in reset button that will revert the system back into its original state before the anomaly was injected.

5.1.1. High Level Overview

The injection software shall catch messages from the software bus and parse them to execute the chosen anomaly. The development team will decide the message structure. The software bus is utilized to receive anomaly execution messages initiated by the user and execute anomalies in the OSK environment.

5.1.2. Denial of Service Injection

Since COSMOS-CFS communication uses a local connection and never connects to an external router, we cannot legitimately “flood the network” which causes the denial of service.

To simulate a DOS, we will use a network flooding tool to inject hundreds of TCP packets to a specific IP address. In our case, we will use Ruby Script on COSMOS to execute this inject command on our own system. When traffic goes above x, we will send a command to disconnect the socket/communication for COSMOS-CFS. (if this is possible from the ground side)

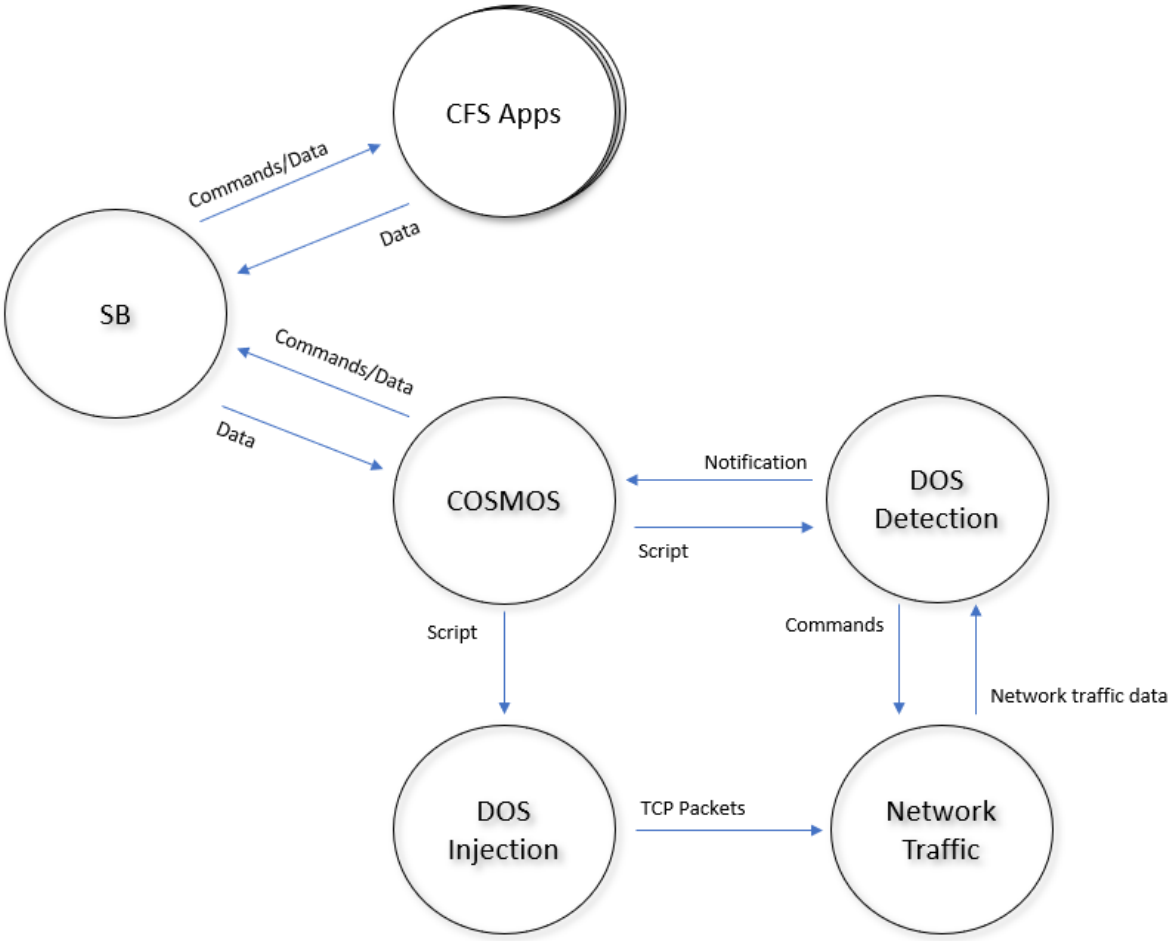


Figure 5 - 1 Denial of Service Injection

5.1.3. Invalid Command Sequence Injection

I. Modify 42 code
II. Create duplicate command
III. Modify command to change order
IV. Code change sends incorrect command to 42 *from COSMOS
V. 42 unable to process command
VI. Indicates error message to COSMOS

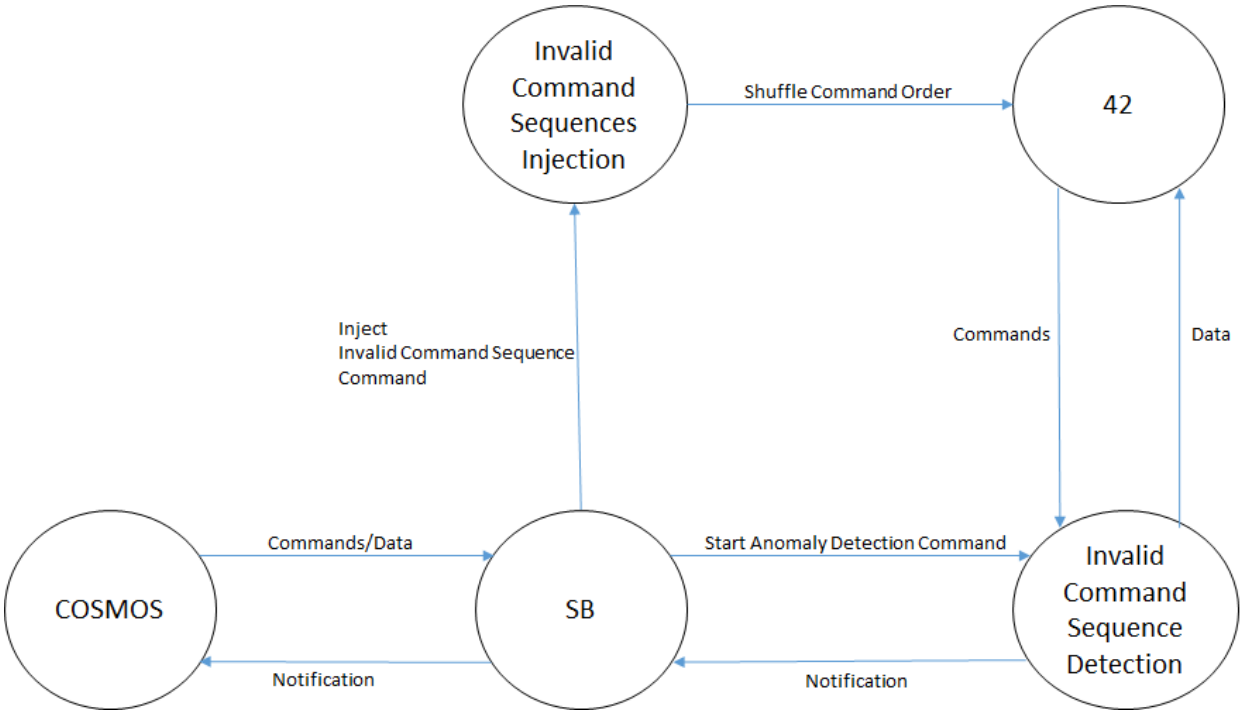


Figure 5 - 2 Invalid Command Sequence Injection

5.1.4. Memory Leak Injection

I. Create a dummy app that is allocated to a large section of memory
II. App gets assigned to CFs
III. Telemetry data receives CFs function and loads onto software bus
IV. Function is then injected with an expected error output

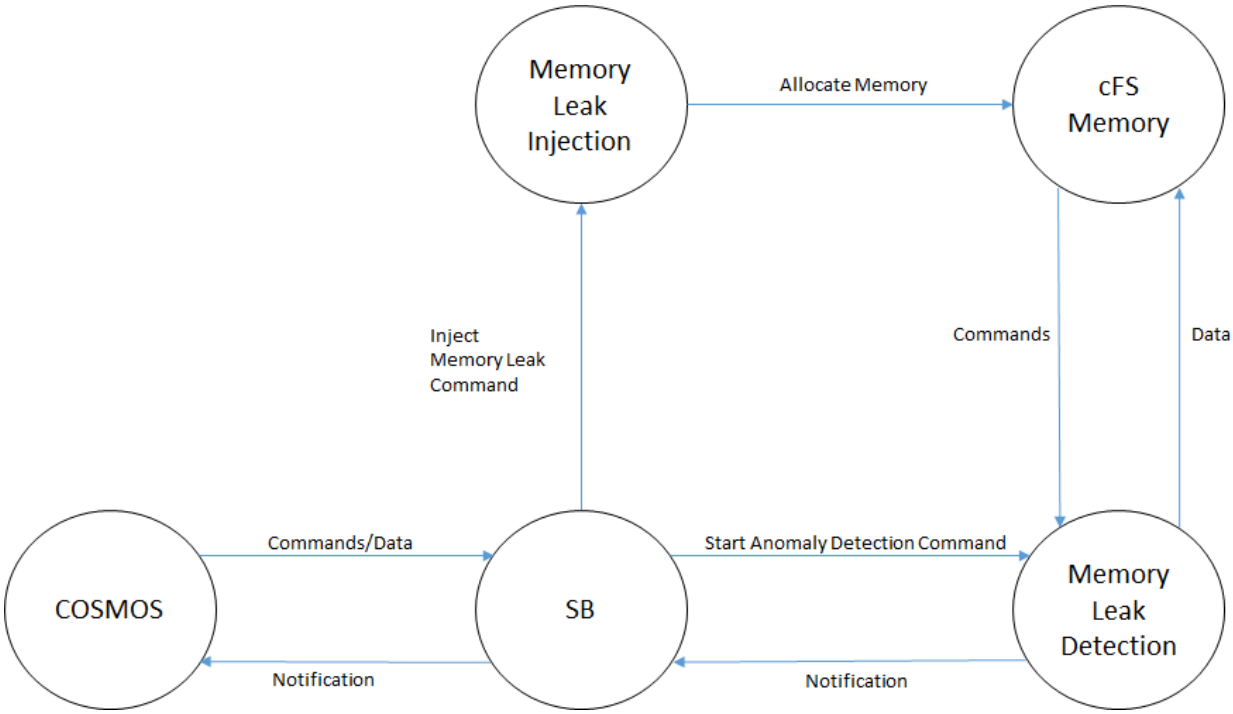


Figure 5 - 3 Memory Leak Injection

5.1.5. Runaway Tasks Injection

I. Spawn orphan thread that dynamically allocates memory
II. Function injected into CFs
III. Telemetry data pushes function onto software bus
IV. Function is then injected with an expected increase in slow time

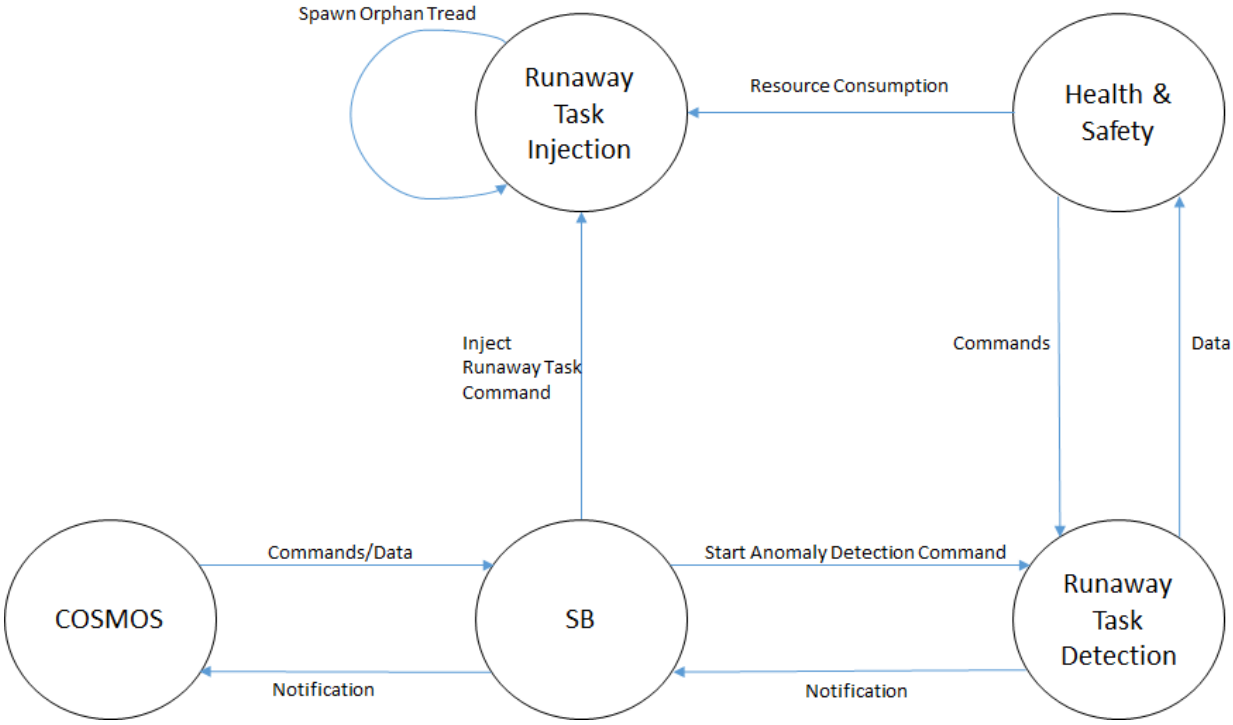


Figure 5 - 4 Runaway Task Injection

5.1.6. Single Bit Error Injection

I. Create a dummy application (SBEI) and make the “rewritable” memory of the satellite.
II. Use the core Flight Executive (cFE) API to read and write the memory into SBEI.
III. Read the memory of SBEI
IV. Flip one bit of SBEI by using an XOR operator
V. Use the flipped bit and write that back into SBEI

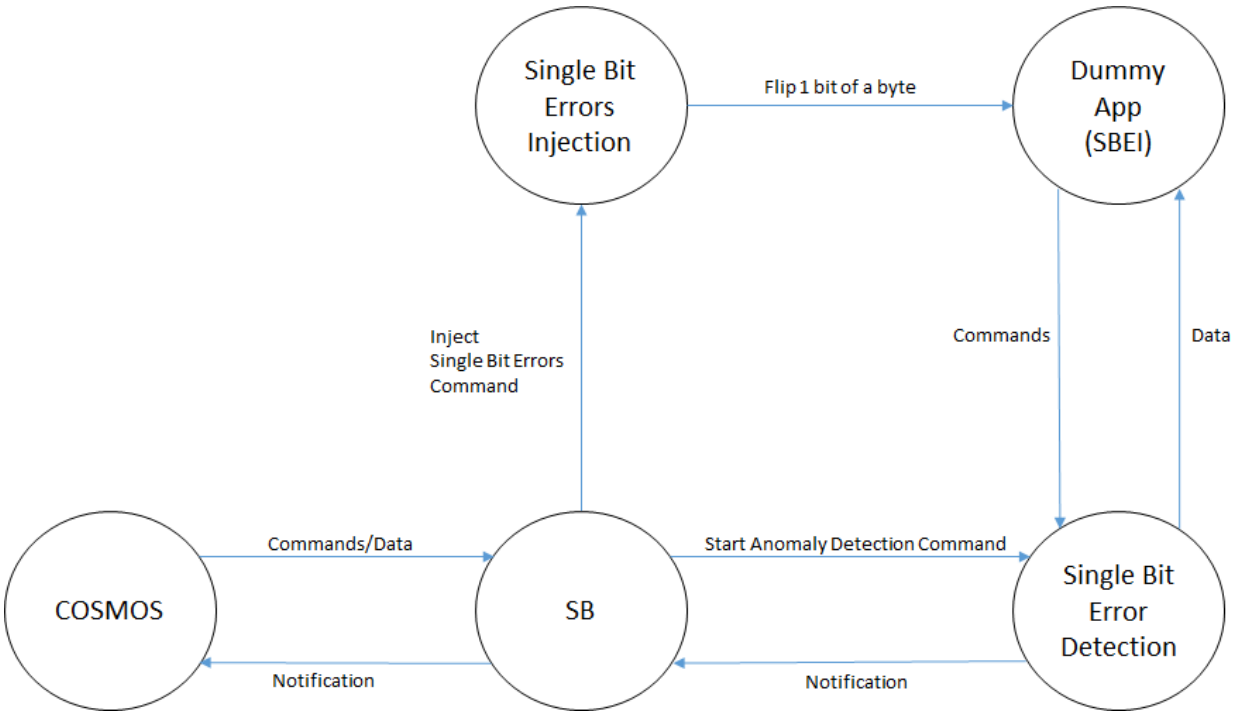


Figure 5-5 Single Bit Error Injection

5.2. Onboard Anomaly Detection

The onboard anomaly detection shall be responsible for the detection of the following anomalies:

- Single bit errors
- Memory leaks
- Runaway tasks

This system shall also alert the user of the presence of any of the above anomalies. The onboard anomaly detection system shall interact with other critical applications using the software bus API (see section 6.3) offered by OSK. All three anomalies will have different methods of detection and will always be running in the background. Onboard anomaly detection processes are intended to be lightweight to consume minimal computational resources onboard the spacecraft. Anomalies such as invalid command sequences and denial of service are more computationally intensive with the abnormalities mentioned above and are on the ground system (Section 5.3).

5.2.1. High Level Overview

The onboard anomaly detection cFS application shall be a background process that checks for anomalies within the system. Onboard detection shall use the “cfe_sb.h” header file to interact with the software bus.

5.2.2. Single bit error Onboard Detection

The single-bit error anomaly detection algorithm shall periodically run the checksum function on the region of memory targeted for single-bit errors. The value returned by the checksum function shall be compared against the latest call to checksum. If there is a discrepancy, an error will be thrown. Otherwise, the algorithm continues as expected, continuously running and verifying the checksum against its last call.

I. Call Checksum to all memory tables
II. If checksum returns 1, continue
III. Else if checksum returns 2, trigger Anomaly notification

5.2.3. Memory Leak Onboard Detection

Memory leaks will be handled by using the memory manager app. The memory manager cFS application is capable of detecting memory leaks when pointed to a region of memory. The injection algorithm will dynamically allocate several bytes without calling the free function, thus creating a memory leak. The process responsible for detecting memory leaks shall periodically call the memory manager and verify that there are no memory leaks in the region of memory targeted for memory leaks.

I. Memory Leak Detection
II. All data storage tables are monitored
III. Code runs to detect error count
IV. If error count exceeds 10, ping anomaly
V. Else, continue running

5.2.4. Runaway Task Onboard Detection

Runaway tasks shall be handled by using the health and safety app. The detection app shall communicate with the health and safety app to monitor computational resource consumption by apps. Injected runaway tasks shall be designed to consume ample resources, which will be caught by the health and safety cFS app, and feedback will be sent to the detection app. The detection app will subsequently notify the user of the runaway task.

I. All data storage tables are monitored
II. Utilize Health and Safety applications to check resource consumption.
III. Review all resource allocations
IV. If resource exceeds overload consumption, ping anomaly
V. Else, keep running

5.3. Ground Based Anomaly Detection

The ground-based anomaly detection algorithm is for the following anomalies:

- Denial of service
- Invalid command sequences

These two anomalies have been selected for ground-based detection to ease onboard cFS computation and improve the detection system's functionality as the denial of service. Invalid commands closer to the ground system than the previous anomalies. The API used is the software bus API for inter-application communication and the time services API for denial of service time thresholds.

5.3.1. High Level Overview

The detection software shall be implemented alongside the COSMOS apps and shall act as an intermediary between the ground base and satellite communications. Denial of service shall use the time services API to detect abnormal request-response cycle times for each request. It will also detect the UDP socket's status and throw an error if it becomes overloaded with requests or disconnects from cFS aboard the satellite. Commands will first be sent through the detection app and processed. If the command is possible to execute, the command will be sent to cFS. Otherwise, an error shall be thrown.

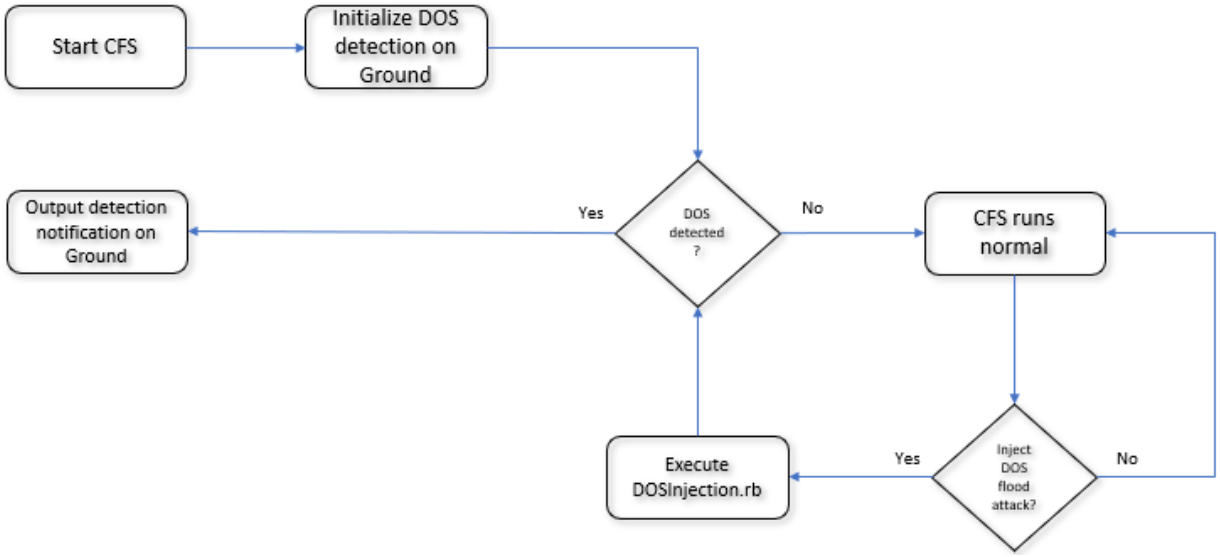
5.3.2. Invalid Command Sequences Ground Detection

Invalid command sequences shall first be passed through the detection application and be processed. The detection application will simulate what will happen after the command runs. If no errors are present after running the command, then the command is sent to the satellite. If an error is present after processing, then an error is thrown, and the user is notified of an invalid command.

I. Run background command sequence detection every time a command is input
II. If command is called on COSMOS
    A. -Is command Successful?
       Yes: Run as intended on 42
       No: Send Anomaly Notification

5.3.3. Denial of Service Ground Detection

There will be a Ruby Script for detection to continuously monitor the network traffic. If the traffic goes above x, it will output a notification of DOS to COSMOS.



6. Requirements Validation and Verification

	6.1.1 Anomaly Injection	
Requirement No.	**Requirement Desciption**	**Verification and Validation Method**
6.1.1.1	The anomaly injection shall inject the following anomalies ● Runaway Tasks ● Memory leaks ● Denial of Service ● Invalid Command Sequence ● Single Bit Errors	
6.1.1.1.1	The runaway task shall create a task in the cFS that does not terminate	TESTED by Runaway Task Injection
6.1.1.1.2	The memory leak shall allocate memory on the cFS without deallocating	
6.1.1.1.3	The denial of service shall spam messages to the software bus	TESTED by DOS Injection
6.1.1.1.4	The invalid command sequence shall send a sequence of commands out of order	
6.1.1.1.5	The single bit error shall flip a bit in the rewritable memory of the cFS	TESTED by Single Bit Error Injection
6.1.1.2	The anomaly injection shall have an automated process that randomly injects an anomaly at random times	
6.1.1.2.1	The automated process shall allow the user to turn it on and off	
6.1.1.3	The anomaly injection shall allow the user to inject their anomaly of choice	TESTED by Runaway Task Injection DOS Injection Single Bit Error Injection

	6.1.2 Onboard Anomaly Detection (OAD)	
Requirement No.	**Requirement Desciption**	**Verification and Validation Method**
6.1.2.1	The OAD shall be an automated process that periodically check the state of the satellite.	
6.1.2.2	The OAD shall detect the following anomalies: ● Runaway Task ● Memory Leak ● Single Bit Errors	
6.1.2.2.1	The runaway task detection shall acquire the CPU data from the cFS	TESTED by Runaway Task Detection
6.1.2.2.1.1	The runaway task detection shall detect anomalous behavior in the CPU data	TESTED by Runaway Task Detection
6.1.2.2.2	The memory leak detection shall acquire the memory state from the cFS	
6.1.2.2.2.1	The memory leak detection shall detect anomalous behaviors in the memory state	
6.1.2.2.3	The single bit error shall acquire the memory data from the cFS	
6.1.2.2.3.1	The single bit error shall detect anomalous behaviors in the memory data	
6.1.2.2	The OAD shall create an event log once an anomaly has been detected	TESTED by Runaway Task Detection
6.1.2.3	The OAD shall create a notification once an anomaly has been detected	TESTED by Runaway Task Detection
6.1.2.4	The OAD shall send the event log and notification next contact with ground	TESTED by Runaway Task Detection

	6.1.3 Ground Base Anomaly Detection (GBAD)	
Requirement No.	**Requirement Desciption**	**Verification and Validation Method**
6.1.3.1	The GBAD shall be an automated process that will constantly check the state of the telemetry data.	
6.1.3.1	The GBAD shall detect the following anomalies: ● Denial of Service ● Invalid Command Sequences	
6.1.3.1.1	The denial of service detection shall acquire the connection time between COSMOS and the cFS	TESTED by DOS Detection
6.1.3.1.1.1	The denial of service detection shall detect anomalous behaviors in the connection time	TESTED by DOS Detection
6.1.3.1.2	The invalid command sequence detection shall acquire the command sequence being sent up to the cFS	
6.1.3.1.2.1	The invalid command sequence detection shall detect anomalous behaviors in the command sequence	
6.1.3.2	The GBAD shall create an event log once an anomaly has been detected	TESTED by DOS Detection
6.1.3.3	The GBAD shall create a notification once an anomaly has been detected	TESTED by DOS Detection
6.1.3.4	The GBAD shall send an event log and notification back to ground	TESTED by DOS Detection

7. Glossary

API - Application Programming Interface
cFE - Core Flight Executive
cFS - Core Flight System
GBAD - Ground Base Anomaly Detection
OAD - Onboard Anomaly Detection
OSK - OpenSatKit
SAID - Satellite Anomaly Injection and Detection
SB - Software Bus

8. References

SRS: https://drive.google.com/file/d/1_a6FILgQFL8icp_bcoqSj2wQlJuggsUW/view

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

COSMOS: https://cosmosc2.com/

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