Preliminary Design Review

for

Satellite Anomaly Injection & Detection Testbed

Fall 2021

Prepared by Diana Degiacomo, Martha Caldera

CSULA / The Aerospace Corporation

November 19, 2021

Table of Contents

1. Overview

1.1. Purpose

The purpose of this document is to consolidate and recapitulate all of the developing information and progress regarding the SAID Testbed, update on progress towards requirements, and snapshot a living schedule/timeline of project progress.

2. Requirements Progress

2.1. Development and Testbed Environment

Provide a flight software, ground station, and spacecraft simulator testbed environment for developing anomaly injection, detection, and resolution capabilities. Team shall establish a development environment with OSK installed and configured to be used for learning about OSK and as a testbed to test anomaly injection, detection, and resolution.

2.1.1. Outcome Overview

All team members have set up an OSK environment and have at least a base understanding of app development from running the Hello World tutorial.

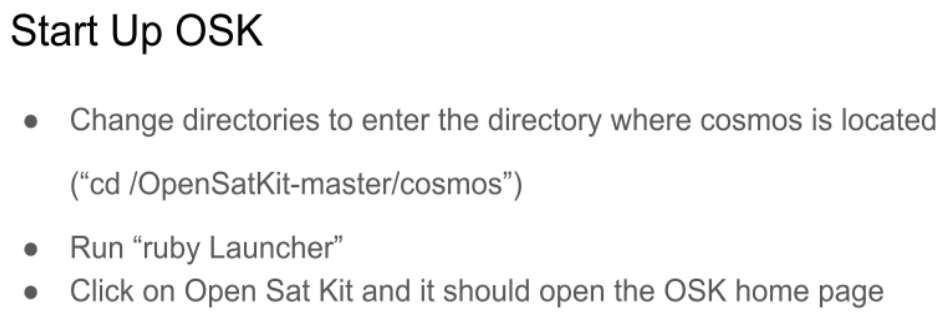

Fig. 2-1 OSK Environment Instructions

Basic Workflow for Hello World

- Green are files created by the program, yellow are files that we will be editing and blue are assumption files and we don't need to worry about them
- When finally starting the cfs server it will run the example application and display in the cfs terminal whether it worked or if there was an error

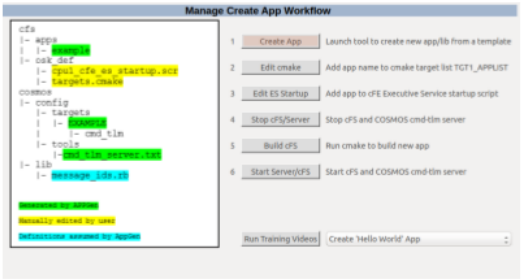

Fig. 2-2 Hello World Tutorial

Command Sender/Terminal

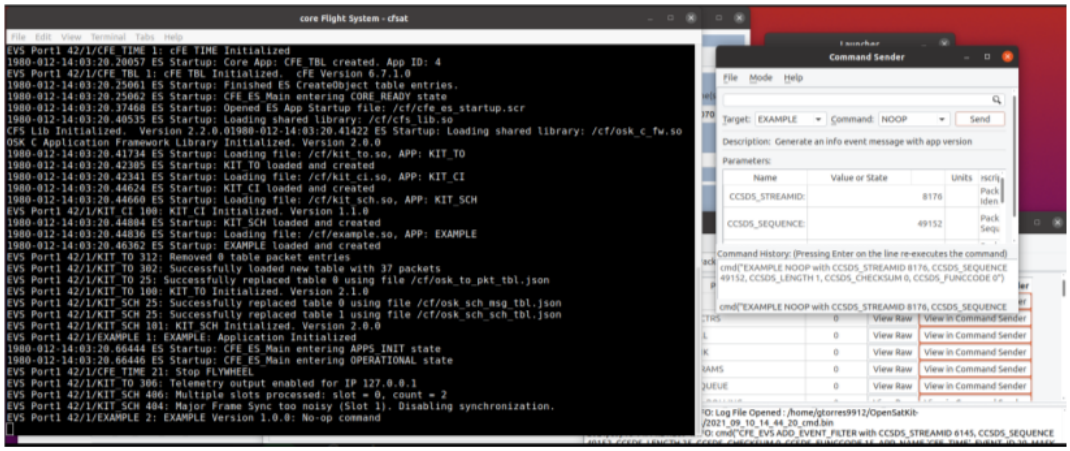

Fig. 2-3 Test Command Sending

2.1.2. Coding Team OSK Image

An image of the coding team's progress is also made available to all team members in our project's shared google drive, titled 'ubuntu OSK1 Clone.ova'. To allow all members to be privy to and have a working environment of the development progress.

2.2. Anomaly Injection

Develop various anomalous scenarios along with a capability to inject these scenarios into the flight system. Team shall develop anomaly inject capabilities to include, but not limited to, the following scenarios:

- component failure (affecting sensor, actuator, thruster, solar panel, thermal, star-tracker, and other systems)
- unexpected halt and/or reboot of main processor
- multiple single-event upsets (SEUs) occurring in short period of time
- loss of communication with ground

2.2.1. Outcome Overview

Our team is focusing on anomaly injection methods explored by the previous year's team. The primary focus being Denial of Service and Single Bit Error.

2.2.2. Denial of Service Injection

Denial of Service injection is fully functioning using to Netwox 76 Synflood tool to overrun the Software Bus of COSMOS successfully denying any transmission of packets.

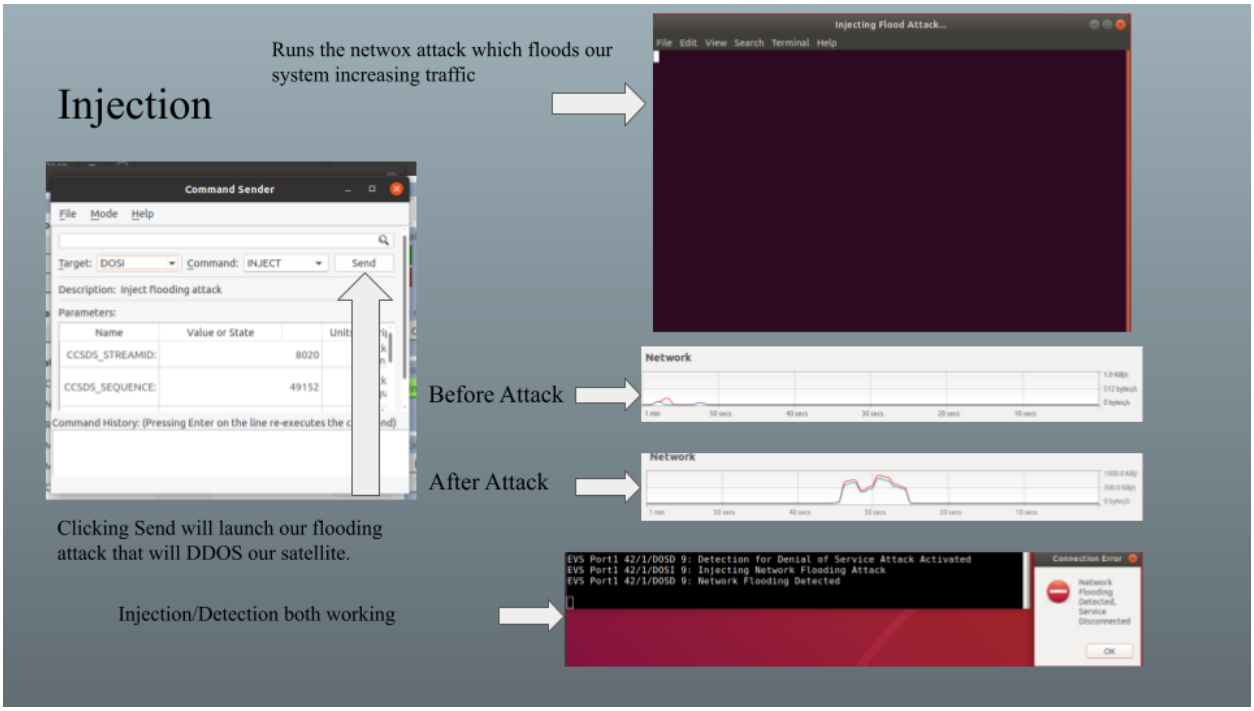

Fig. 2-4 Denial of Service Injection & Detection

2.2.3. Single Bit Error Injection

When the Single Bit Error Injection command is ran, it changes a bit from a byte of memory in the Single Bit Error Injection Application which is stored in the cFS. This will throw off the parity bits which keep track of the even and odds of 1s. The byte is only used as a temporary memory so it isn't stored anywhere in the Core Flight System.

2.3. Anomaly Detection

Develop the capability to detect anomalous behaviors during the mission and generate notifications when anomalies are detected.

2.3.1. Outcome Overview

Areas of focus are Denial of Service and Single Bit Error.

2.3.2. Denial of Service Detection

Denial of Service detection is fully functioning. When the transfer rate is over 400 bytes a Denial of Service attack is detected. The user is notified by a warning message pop-up. See fig. 2-4.

2.3.3. Single Bit Error Detection

Hamming code can pinpoint the location of a single-bit error and correct it. Hamming code makes it so certain bits in the data keep track of the rest of the bits. The single-bit error detection and resolution work hand-in-hand. Once the Hamming code detects an error it can fix it, the downside of this detection and resolution systems is the uncertainty around whether this will work with multiple events. When it comes to multiple bit errors, hamming cannot pinpoint the location of the errors to correct but can still detect that there is an error

2.4. Anomaly Resolution

Develop the capability to automatically resolve anomalies after they have been detected.

2.4.1. Outcome Overview

Our aim with resolution is to ultimately deploy machine learning in the ground system that will automatically resolve on-board and ground anomalies.

2.4.2. Denial of Service Resolution

For Denial of Service, we are going to be using SYN cookies to manage incoming data to prevent half-open connections and manage the queue. This is to prevent our server from overloading and crashing. When our server's queue is managed and in stable conditions, we plan to use machine learning techniques to validate incoming packets. If these packets are valid according to our machine learning algorithms then we will allow them into the queue.

2.4.3 Single Bit Error Resolution

For Single Bit Error resolution, our team is using Hamming code which makes it so certain bits in the data keep track of the rest of the bits. These specific bits will be a 1 when there's an odd number of 1s in the section of bits they're checking and a 0 where there's an even number of 1s. After creating this new string of bits we will then store the indices of the location where there is a 1 in the new string of bits. We will convert the indices to binary and conduct the XOR function and the result should always be 0000 if there is no error. So, when the injection command is ran it will flip a bit in the data which will throw off the parity bits which keep track of the even and odds of 1s. When we run the XOR function on the new string of bits it will not return 0000 and instead, return the binary location of where the error is located which we will then convert to an integer and flip the bit at that location of the string resolving the issue.

3. Timeline

3.1. Living Document

Timeline - Living Document Link

3.2. Current Timeline Snapshot

Fall																	
Week Date	23-Aug	30-Aug	6-Sep	13-Sep	20-Sep	27-Sep	4-Oct	11-Oct	18-Oct	25-Oct	1-Nov	8-Nov	15-Nov	22-Nov	29-Nov	6-Dec	13-Dec
Week #	1	2	3	4	5	6	7	8	9	10	11	12	13	Thanksgiving Week	14	15	Finals
Meeting Topic	Intro: Kickoff Meeting	-Liaison Meeting -Get familiar with & install OSK	-Liaison Meeting -Group1: Launch attack research -Group2: Detect DoS research -Group3: Run simulator	Status update & continue working on group assignments	Liaison Update on **9/25** and 2021-2022 Requirements received	New Groups: Group1: Run Simulator Group2: Explore new requirements & create timeline	-Start of Doc. Phase -New Groups: Group1: Uplink & Downlink Group2: Improve timeline & explore detection	-Liaison Update on **10/15** -Group1: Continue Uplink & Downlink -Group2: Continue with detection & doc. phase	-Group1: Finish Uplink & Downlink, Begin Resolutions -Group2: Finish Detection, Begin SRS doc.	-Group1: Continue Resolutions -Group2: Finish SRS doc.	-Liaison Update on **11/5** -Group1: Finish Resolutions -Group 2: Begin PDR & PMR doc.	-New groups: Coding Group: Documentation Group: Finish PDR & PMR	Liaison Update on **11/19**		Senior Design Presentation on **12/3**	-Liaison Meeting on **12/10** -End of Doc. Phase	
Milestone			OSK Test Bed Requirement Met		Group assignments & presentations complete		-Simulation ran successfully -Timeline created			-DoS Injection & Detection -SBE Injection							
Winter																	
Week Date	20-Dec	27-Dec	3-Jan	10-Jan	17-Jan												
Week #	Break	Break	16	17	18												
Meeting Topic			Start of Code Phase	Start of Testing Phase													
Milestone																	
Spring																	
Week Date	24-Jan	31-Jan	7-Feb	14-Feb	21-Feb	28-Feb	7-Mar	14-Mar	21-Mar	28-Mar	4-Apr	11-Apr	18-Apr	25-Apr	2-May	9-May	16-May
Week #	19	20	21	22	23	24	25	26	27	Spring Break	28	29	30	31	32	33	Finals
Meeting Topic	Liaison Update on **1/28**			Liaison Update on **2/18**			Liaison Update on **3/11**	TRR		-End of Code/Testing Phase -Start of Wrap Up of Phase	Liaison Update on **4/08**		End of Wrap Up Phase	Liaison Update on **4/29**		Presentation at Aerospace	
Milestone																	

Fig. 3-1 Timeline Snapshot

4. Additional Information

4.1. Ground System & Machine Learning Implementation

Our goal for resolution implementation is machine learning on the ground system for both ground and onboard anomalies as COSMOS, the ground system, will have more computing capabilities. Machine learning consists of designing and constructing methods that give computers the ability to learn from past data, without being explicitly programmed, and then make predictions on future data.

4.1.1. Uplink

Command Sender allows communication with the satellite from COSMOS. When a command is sent to the core flight system it sends the message-id from the command and the name of the command. The satellite then communicates with itself and runs a function called CFE_EVS_SendEvent(). The function is what communicates and prints to the core flight system terminal.

4.1.2. Downlink

COSMOS updates the command counter and telemetry view by the satellite running the commands and updating the counters. If an error occurs when running a command then the satellite will let COSMOS know an error occurred and create a file that is stored in COSMOS.

4.1.3. Ground System

COSMOS implements a client-server architecture with the Command and Telemetry Server and the various other tools typically acting as clients to retrieve data. The Command and Telemetry Server connects to the targets and sends commands and receives telemetry (status data) from them. Automation of resolution will utilize machine learning to apply the correct solution for onboard anomalies once telemetry data is received. Then a solution can be implemented to onboard the satellite.