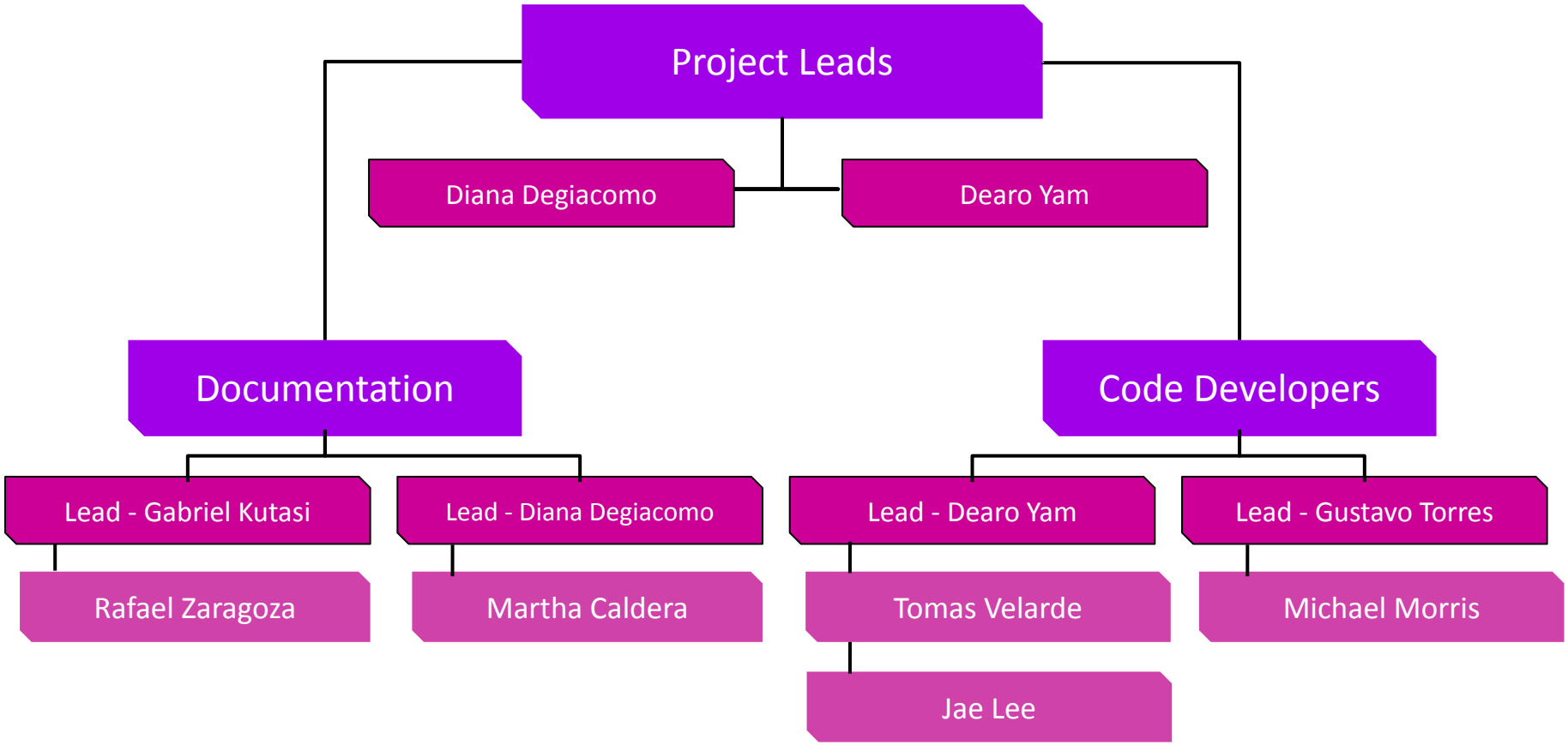
Satellite Anomaly Inject & Detection(SAID) Testbed

Powered by CSULA & Aerospace Corporation
Advisor: Zilong Ye

December 3, 2021

CONTENTS

Organizational Chart - Diana



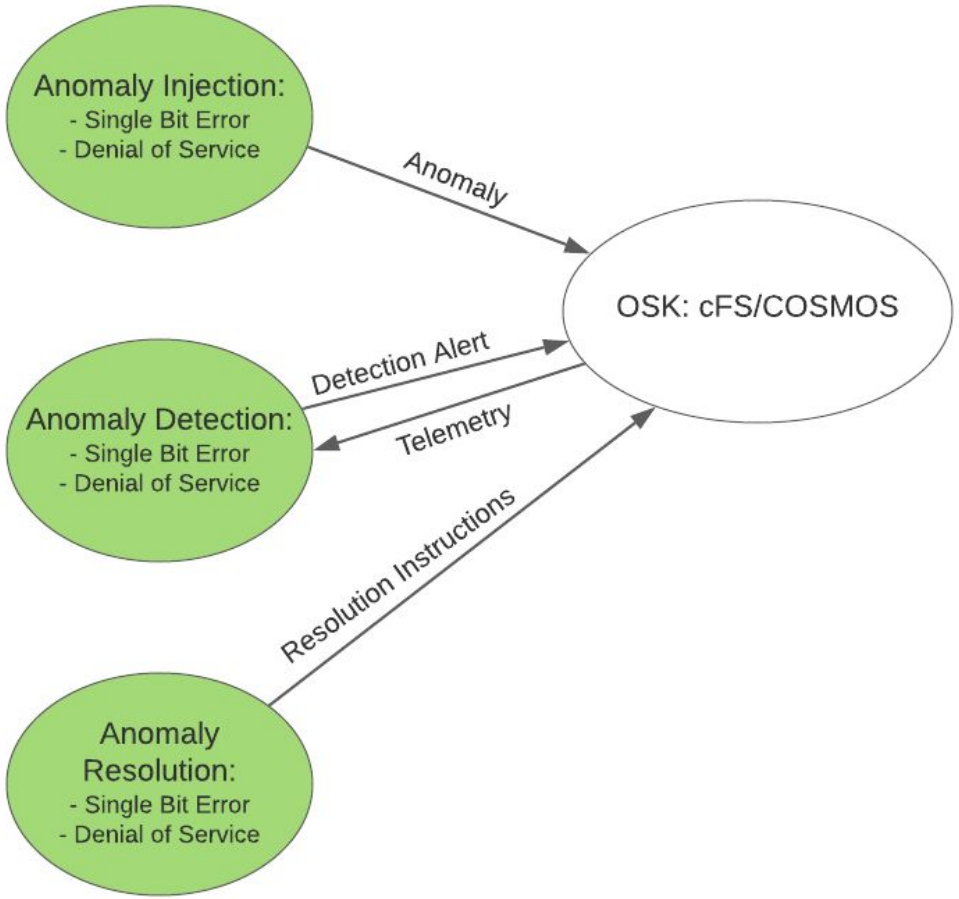
Welcome Agenda - Diana

- Project Introduction - Diana Degiacomo
- OpenSatKit - Tomas Velarde
- COSMOS - Jae Lee
- Current Version of Denial of Service - Dearo Yam
- Current Version of Single Bit Error - Gabriel Kutasi
- Future Work for Denial of Service - Martha Caldera
- Future Work for Single Bit Error - Michael Morris
- Milestones - Rafael Zaragoza
- Challenges - Gustavo Torres
- Questions - Gustavo Torres

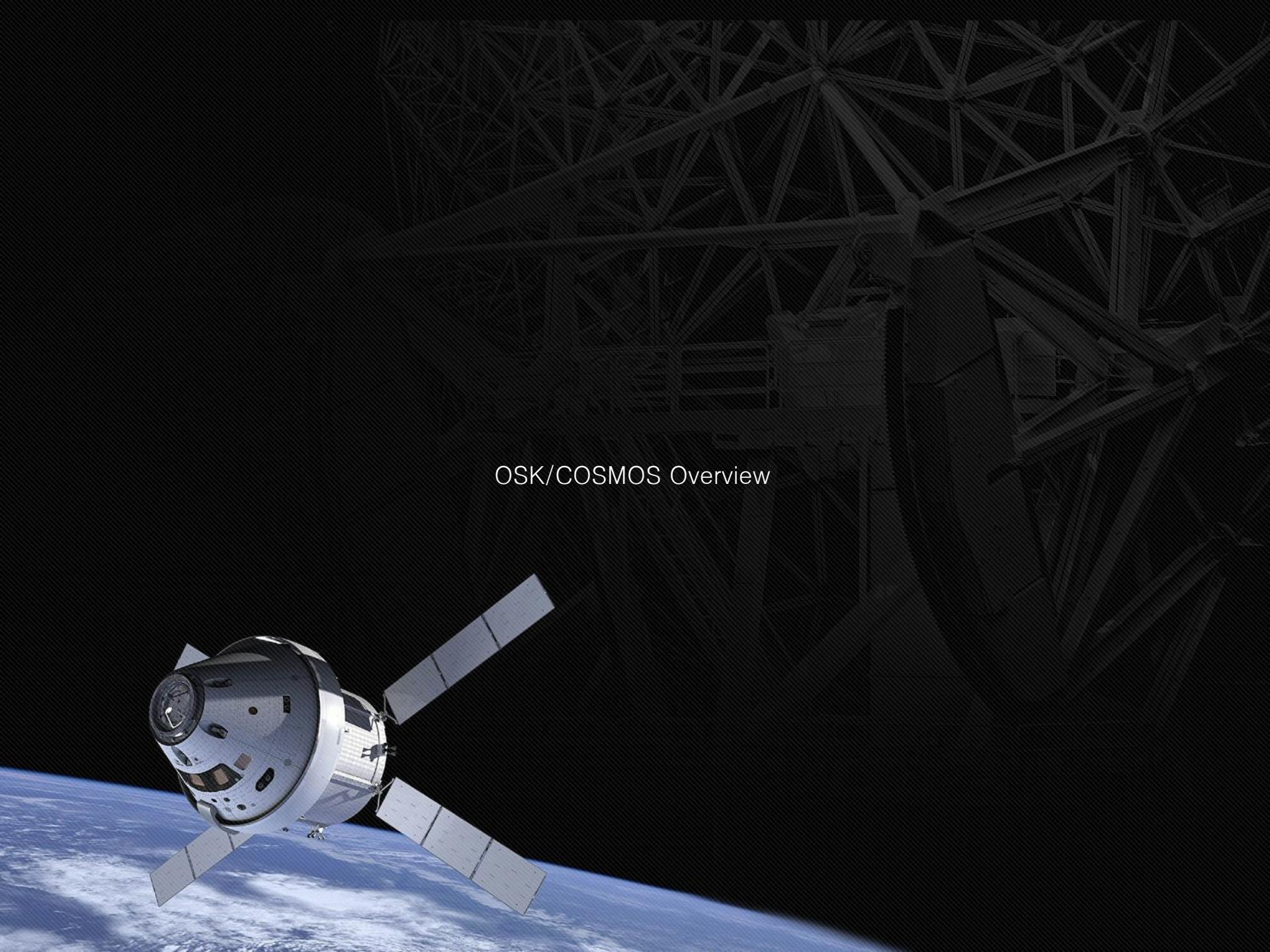
Project Introduction - Diana

Overall Project Goal: Protect satellites against malicious attacks

- Industry Open Source Software: OSK - A core Flight System (cFS) Platform
  - Build apps to interface with this platform:
    - Inject Anomalies
    - Detect Anomalies
    - Resolve Anomalies
- Anomalies:
  - Single Bit Error
  - Denial of Service



OSK/COSMOS Overview

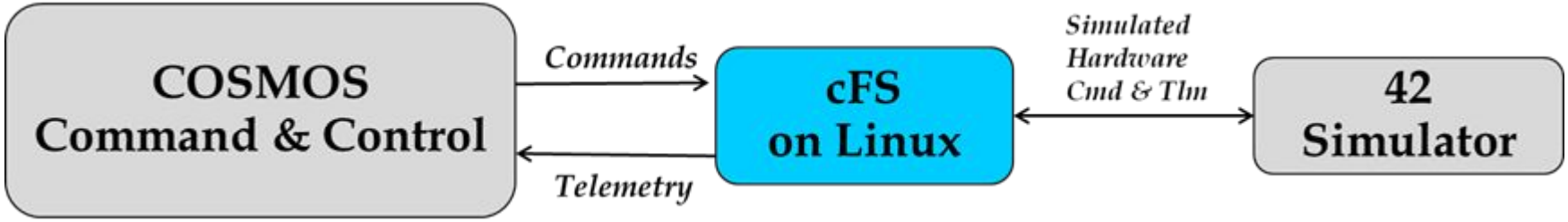


OSK - Tomas Velarde

- **Open Satellite Kit (OSK) -** A satellite simulation environment.

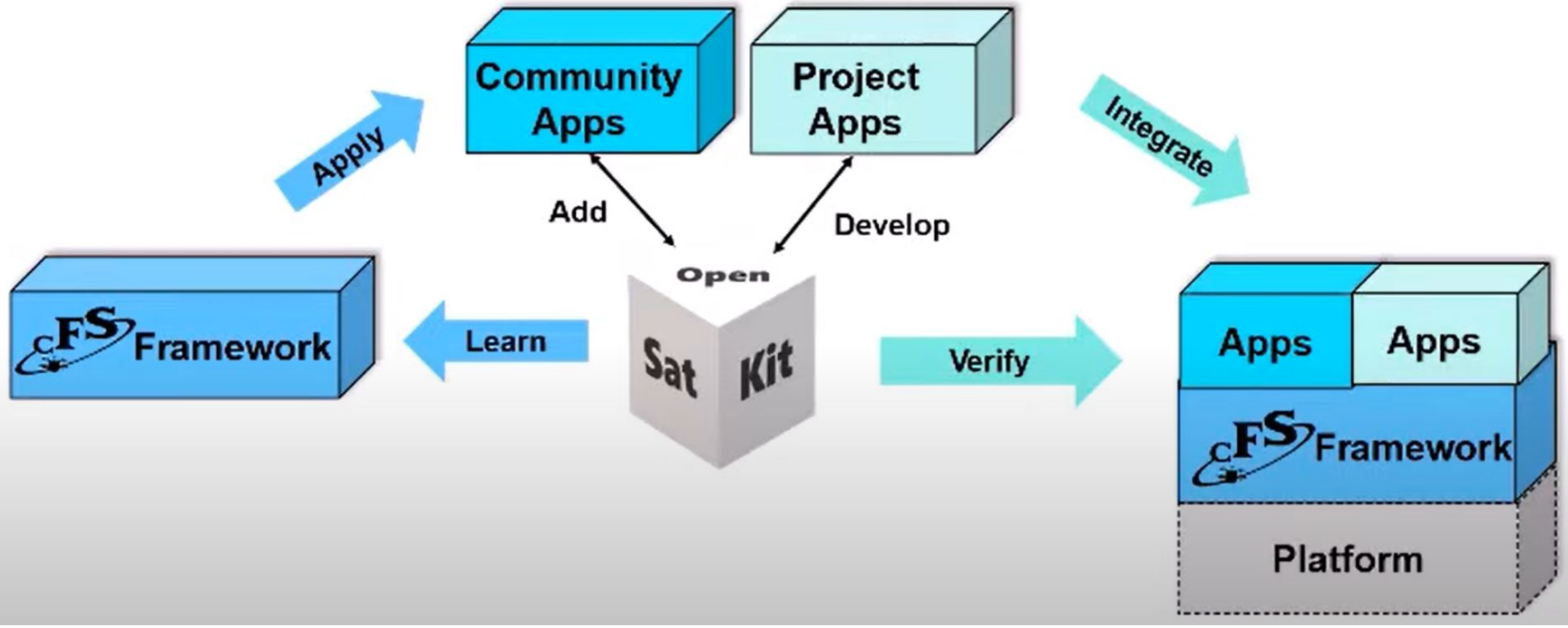
- **Our Team's Goal with OSK-** To use this simulation environment to simulate attacks against satellites and satellite malfunctions to produce effective countermeasures and remedies.

The 3 main components of Open Satellite Kit:



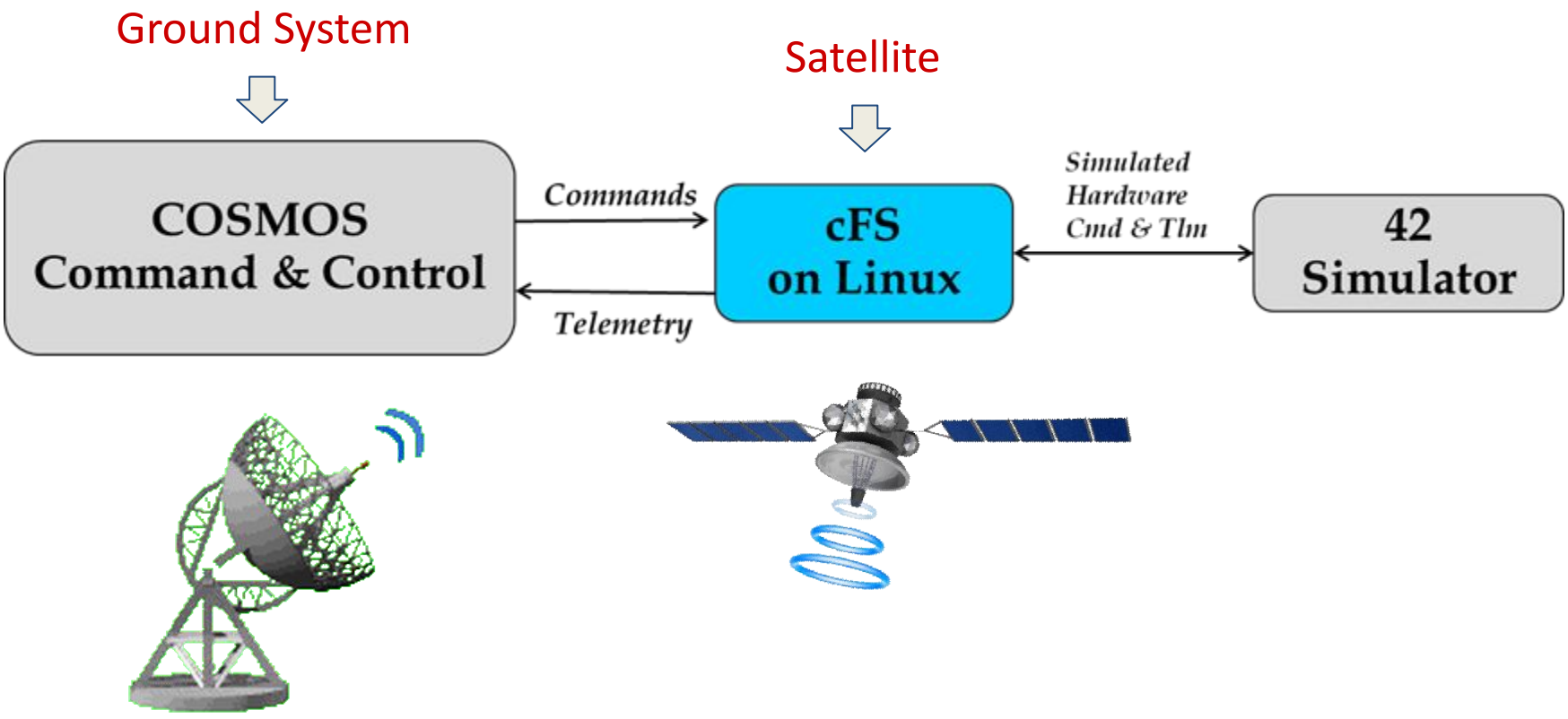
OSK - Tomas Velarde

- Purpose of OSK



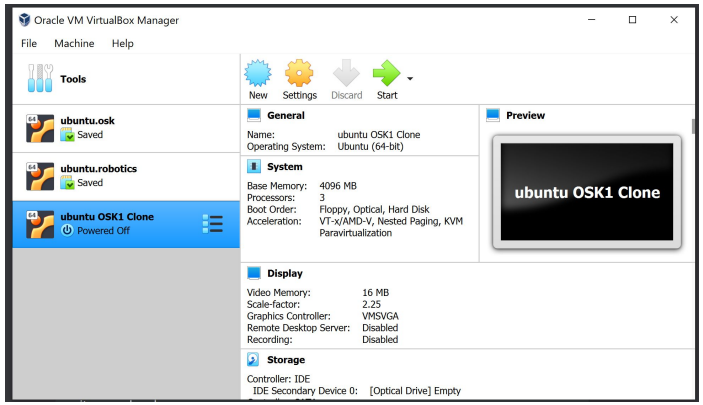
OSK - Tomas Velarde

Open Satellite Kit Composition: Three open source tools

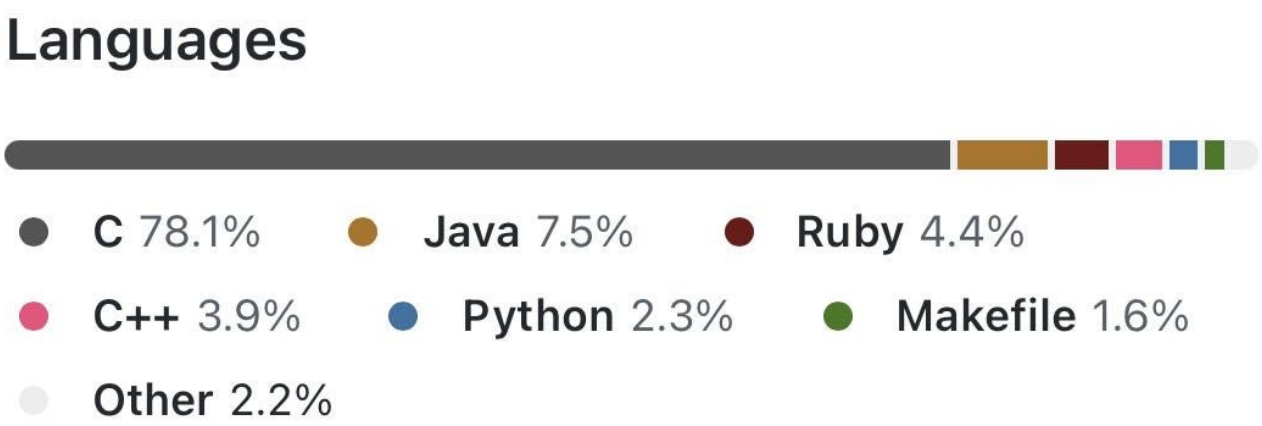


OSK - Tomas Velarde

- **Open Satellite Kit runs on Ubuntu Linux**
  - Have installed Oracle VM VirtualBox, a virtual machine set up with Ubuntu Linux to run



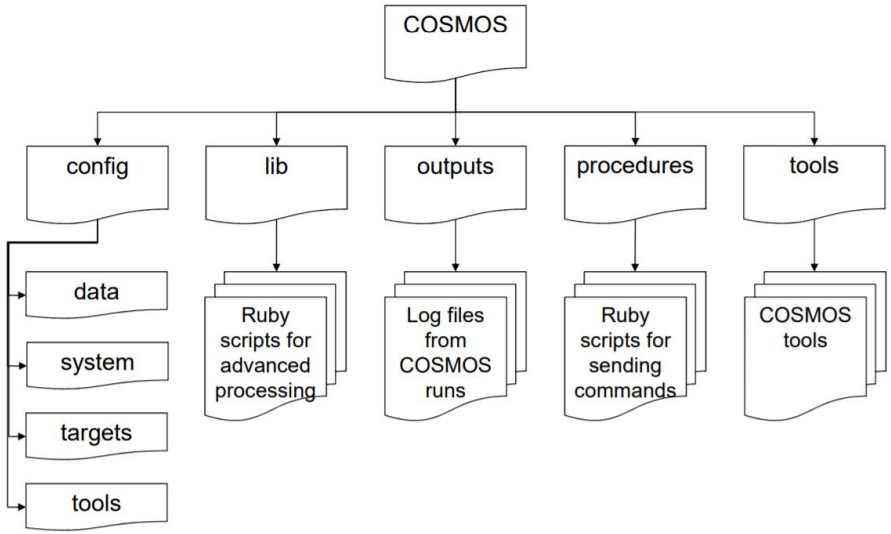
- **Programming languages used in Open Satellite Kit:**

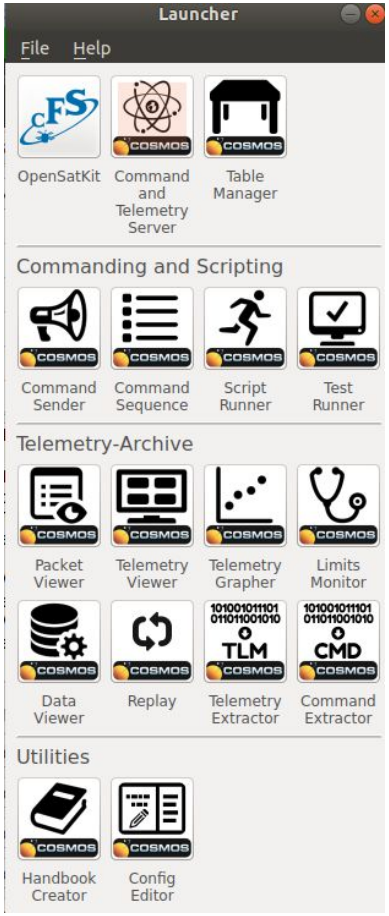


Cosmos - Jae Lee

- *What is COSMOS?*
  - A command and control platform
  - Open-source ground system
  - The primary OSK user interface
- *What is it used for?*
  - To communicate with the satellite
  - Monitor the performance and health
  - Display the data
- *What is it used with?*
  - Test equipment
  - Development boards
  - Satellites

Navigating COSMOS - Jae Lee

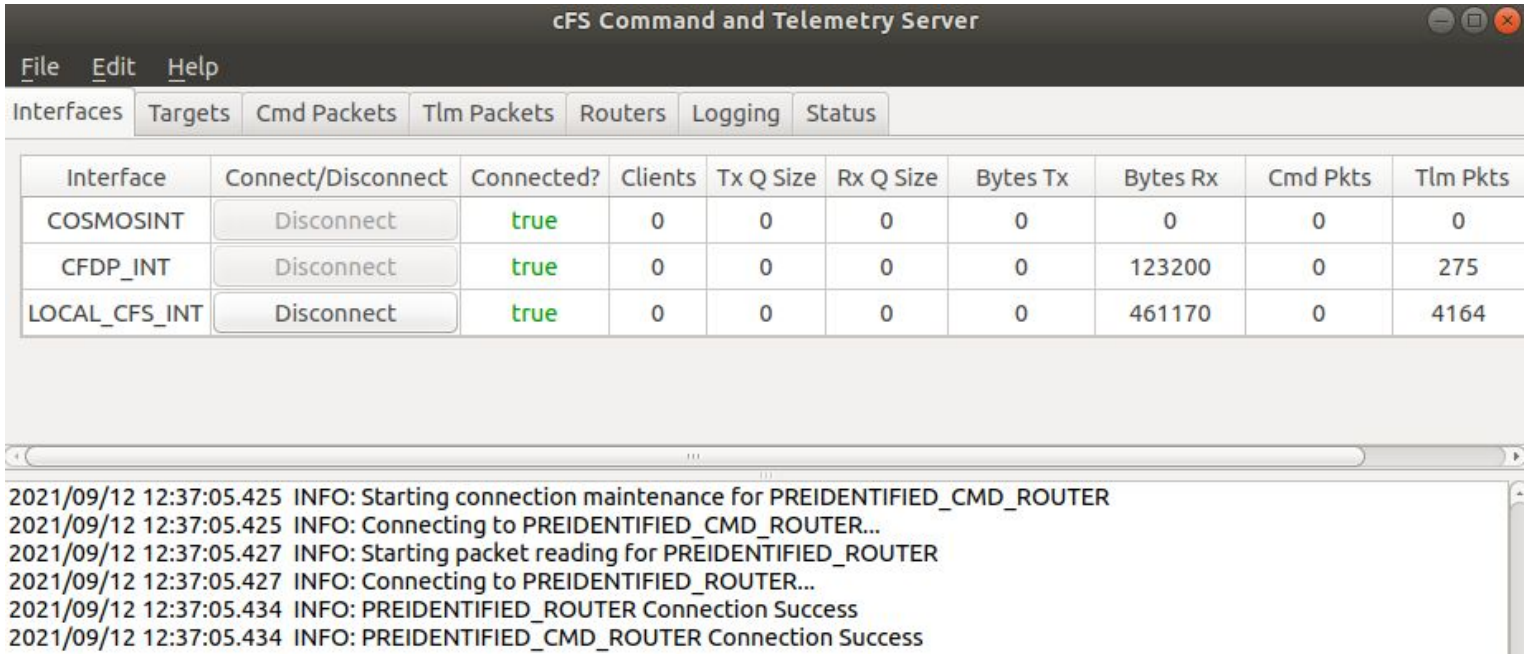




- Target
  - Destination for commands
- Tools
  - The Launcher screen consists of tools

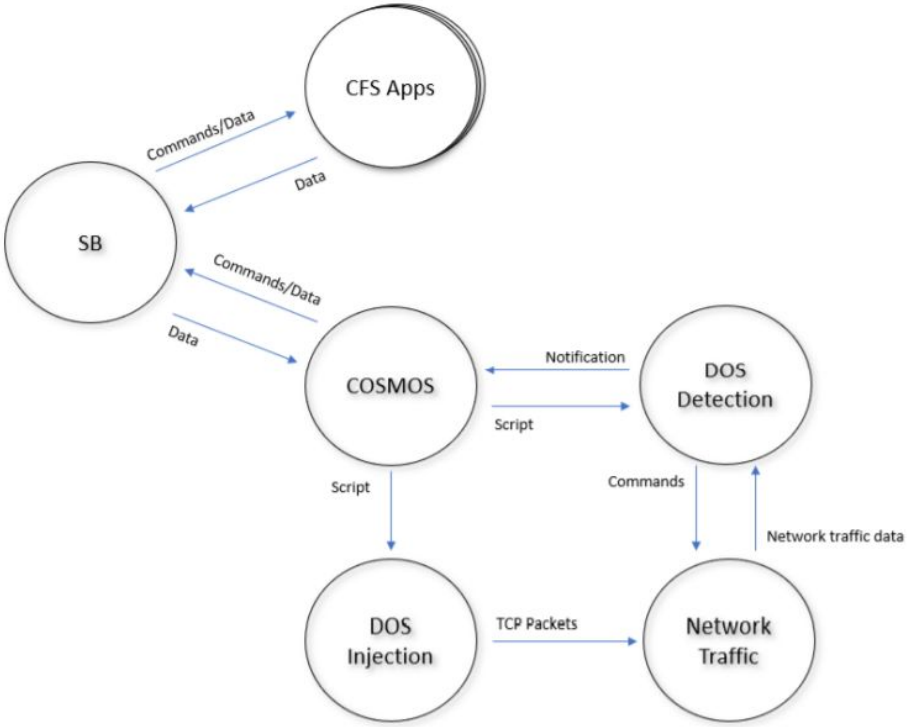
COSMOS Tool - Jae Lee

- *Command & Telemetry server tool*
  - Overview of all the connections in the system



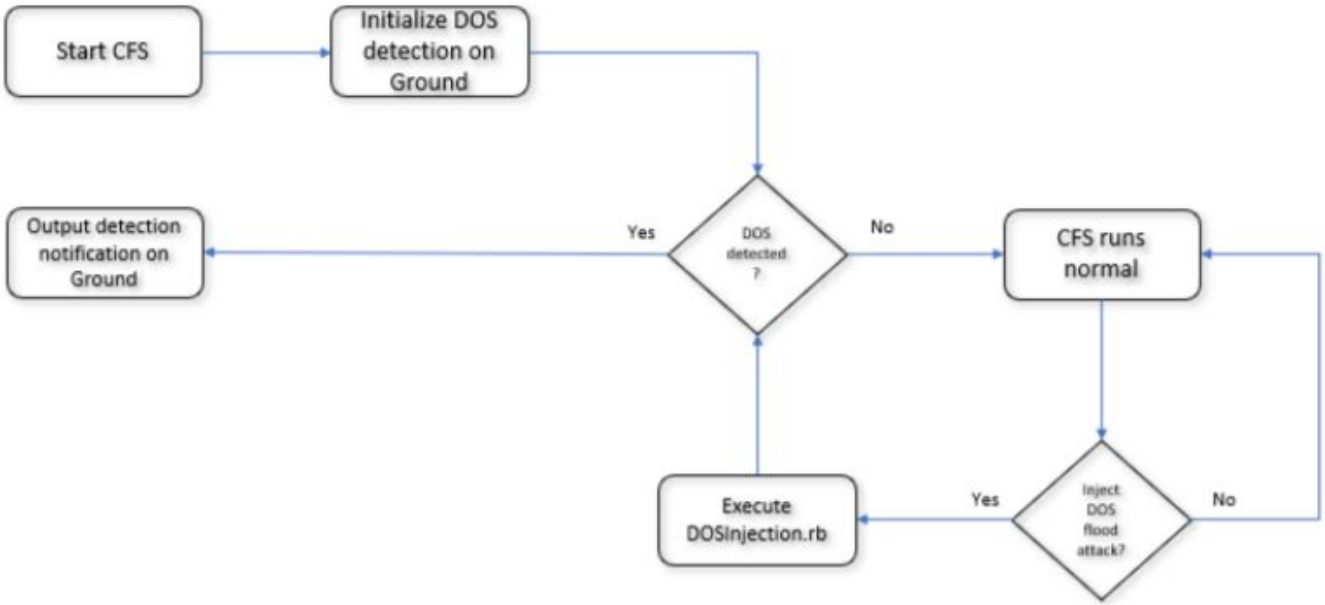
Using COSMOS - Jae Lee

- *Denial of Service (DOS) Injection*



Using COSMOS (cont.) - Jae Lee

- *Denial of Service (DOS) Detection*



Current Work & Future Goals

Current Model Of DDOS - Dearo

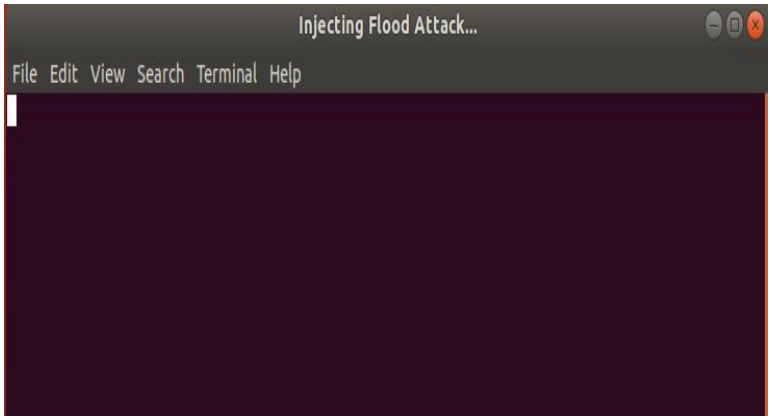
- *Injection*
  - *injects a syn flood attack*
  - *happens for 8 seconds*

```
char command1[] = "gnome-terminal --title='Injecting Flood Attack...' -e 'sudo timeout 8s netwox 76 -i ";
char command2[] = " -p 23 -s raw' >/dev/null 2>&1 &";
FILE *ip = popen("hostname -I", "r");
fscanf(ip, "%s", ipString);
snprintf(command, 150, "%s%s%s", command1, ipString, command2);
//printf("%s", command);
system(command);
```

Step1 ● *Demonstration*

"Click"

Final:



Current Model Of DDOS (cont.) - Dearo

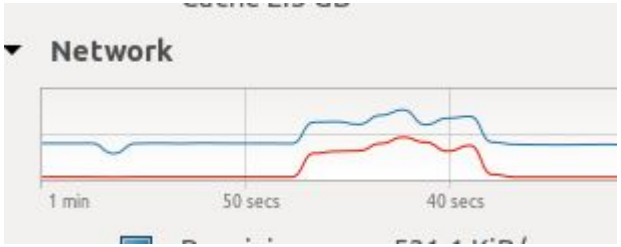
- *Before Attack*

```
Tcp:
    11 active connection openings
    1 passive connection openings
    4 failed connection attempts
    0 connection resets received
    5 connections established
    848 segments received
    797 segments sent out
    4 segments retransmitted
    0 bad segments received
    7 resets sent
```

- *After Attack*

```
Tcp:
    13 active connection openings
    2 passive connection openings
    4 failed connection attempts
    1 connection resets received
    4 connections established
    185135 segments received
    117371 segments sent out
    4 segments retransmitted
    0 bad segments received
    115728 resets sent
```

- During Attack

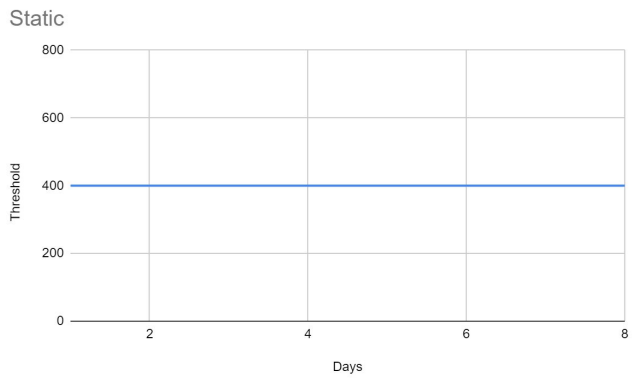


Current Model Of DDOS (cont.) - Dearo

- *Detection Of DOS*

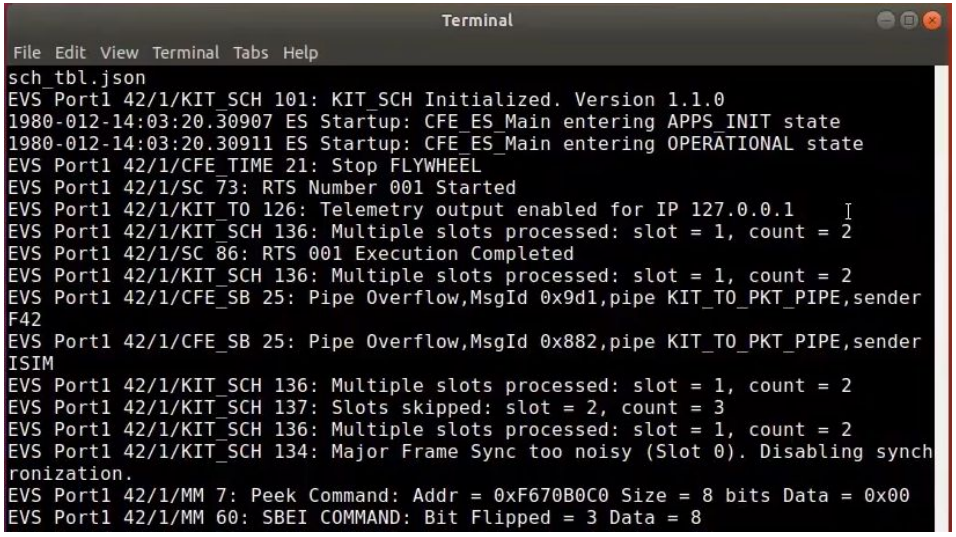
```
file1 = fopen("/sys/class/net/enp0s3/statistics/rx_bytes", "r");
fscanf(file1, "%d\n", &initByte);

sleep(1);

file2 = fopen("/sys/class/net/enp0s3/statistics/rx_bytes", "r");
fscanf(file2, "%d\n", &byte);

rate = (byte-initByte)/1000.0;
//printf("%d : %d = %f\n", byte, initByte, rate);
//printf("%f\n", rate);
```

```
if(rate > 400.0) {
    CFE_EVS_SendEvent (DOSD_DETECT_INF_EID, CFE_EVS_INFORMATION, "Network Flooding Detected\n");
    detected = true;
```



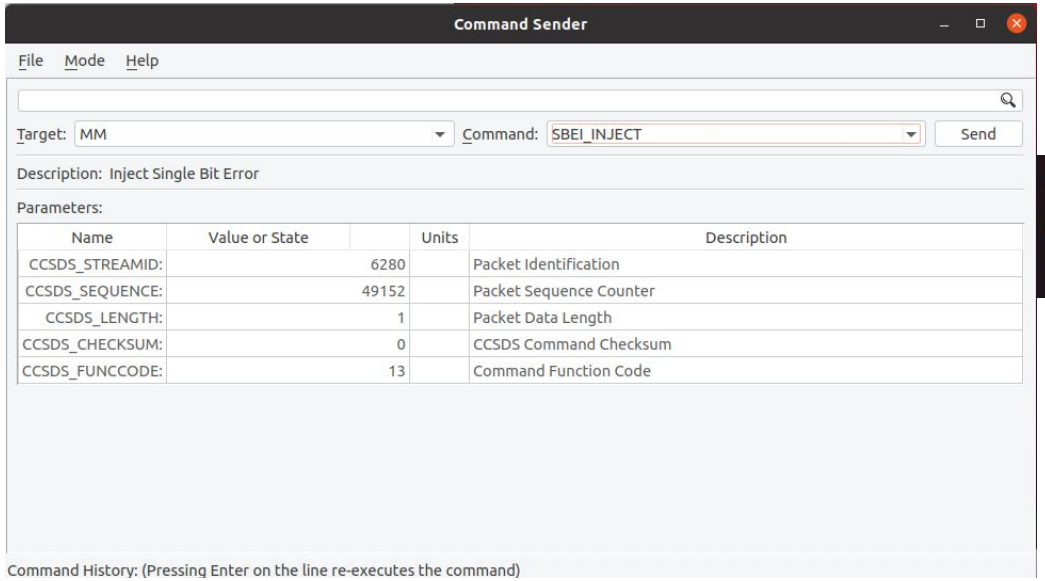
Single Bit Error - Gabe

- *Single Bit Error Injection*
  - *Accessed through Command Sender*
  - *Injection command is located within the memory manager (MM)*
  - *Utilizing PEEK_MEM command will allow the user to view the memory that will be injected with the single bit error command*
  - *SBEI_INJECT will flip a bit in the memory, changing the data stored*



Single Bit Error - Gabe

- *Single Bit Error Injection*
  - *The bits of the application's data is changed when the command is called*
  - *The changed data isn't stored to cosmos or cfs*




```
EVS Port1 42/1/MM 60: SBEI COMMAND: Bit Flipped = 7 Data = 80
EVS Port1 42/1/MM 60: SBEI COMMAND: Bit Flipped = 6 Data = C0
EVS Port1 42/1/MM 60: SBEI COMMAND: Bit Flipped = 6 Data = 80
EVS Port1 42/1/MM 60: SBEI COMMAND: Bit Flipped = 6 Data = C0
```

Future Goals of Denial of Service - Martha

- *Resolution Implementation: Implement machine learning on the ground system for both on **ground** and **onboard** anomalies*
    - *Correction of Denial of Service (DDOS)*
        - *SYN cookies - This is to prevent our server from overloading and crashing*
        - *Machine learning techniques to validate incoming packets*

Future Goals of Single Bit Error - Michael

- *Detection of Single Bit Error (SBE)*
  - *Fully implement Hamming Code*

Adding Parity Bits to 0101 0010 1001 = 0wx1 y010 z010 1001			
Parity #	**Check**	**# of 1's**	**Parity Bit Value**
Parity 1 (w)	_w_1_0_0_0_0_0_1	2	0
Parity 2 (x)	_ _x1_ _10_ _10_ _01	4	0
Parity 4 (y)	_ _ _ _y010_ _ _ _1001	3	1
Parity 8 (z)	_ _ _ _ _ _ _ _z0101001	3	1
Output: 0001 1010 1011 1001			

Future Goals of Single Bit Error - Michael

- *Correction of Single Bit Error (SBE)*
  - *Hamming code can also correct single bit errors*

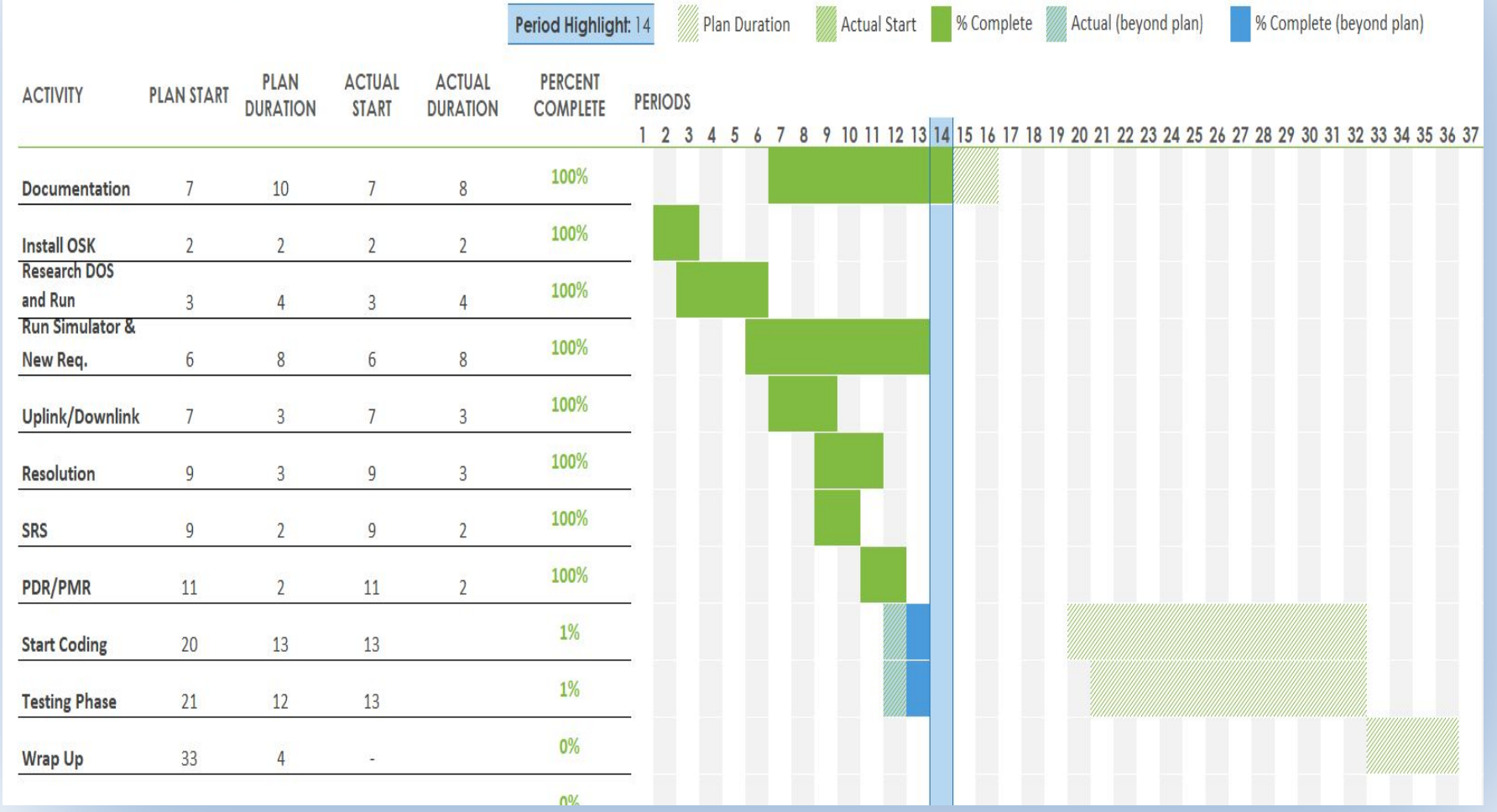
0 0 0000	1 0 0001	2 0 0010	3 1 0011
4 1 0100	5 0 0101	6 1 0110	7 0 0111
8 1 1000	9 0 1001	9 1 1010	11 1 1011
12 1 1100	13 0 1101	14 0 1110	15 1 1111

0011
0100
0110
1000
1010
1011
1100
⊕ 1111
1011

Current Milestones

PMR - Rafael

SAID Planner



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Documentation	7	10	7	8	100%
Install OSK	2	2	2	2	100%
Research DOS and Run	3	4	3	4	100%
Run Simulator & New Req.	6	8	6	8	100%
Uplink/Downlink	7	3	7	3	100%
Resolution	9	3	9	3	100%
SRS	9	2	9	2	100%
PDR/PMR	11	2	11	2	100%
Start Coding	20	13	13		1%
Testing Phase	21	12	13		1%
Wrap Up	33	4	-		0%
					0%

Milestones

- *Installed and familiarized with OSK and developed first HELLO WORLD!*
- *Identified the uplink and downlink process*
- *Learned ML amongst the students*
- *Researched datasets to use for ML for our resolutions*
- *Familiarized what is Denial of Service and Single Bit Error*
- *Developed a resolution for DDOS and Single Bit Error*
- *Finalized Documentation such as PDR, PMR, SRS, Timeline*

Challenges & Questions

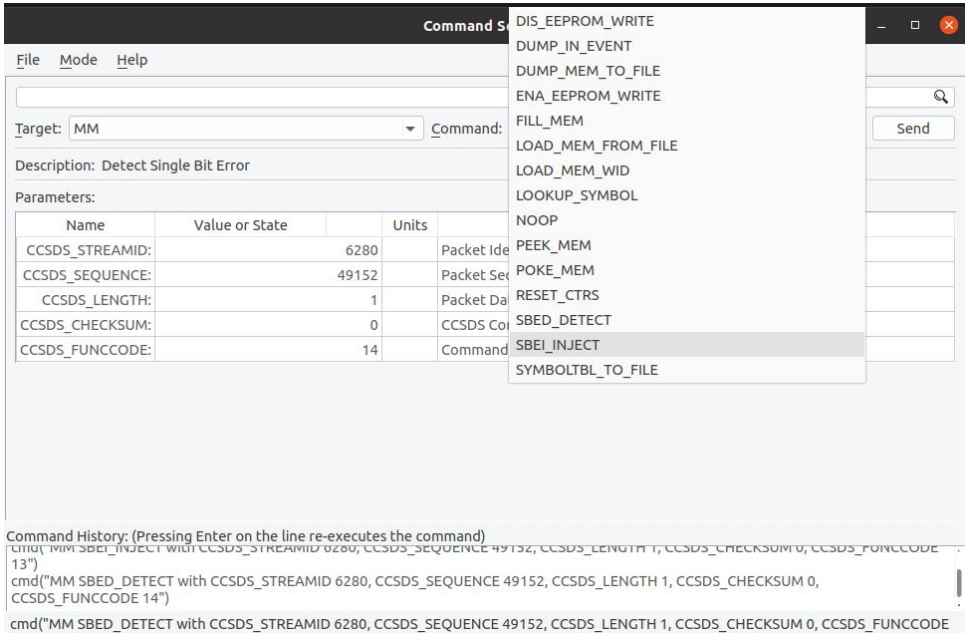
Challenges - Gustavo

Single Bit Error

- *Not able to read the data from an application twice in one run.*
- *Cannot send a command to satellite while it's already running a command from the same application*
- *Having issues importing the CFE_utils library to another application*

Denial of Service

- *Lack of a proper dataset at the start*
- *Having to learn a new topic (machine learning) to some students*



- **Gustavo**

Questions and Comments

THANK YOU