Senior Design Final Report

Satellite Anomaly Injection & Detection Testbed (SAID Testbed)

Version 1.0 05/13/2022

Team Members:

Martha Caldera
Diana Degiacomo
Gabe Kutasi
Jae Lee
Michael Morris
Gustavo Torres
Tomas Velarde
Dearo Yam
Rafael Zaragoza

Faculty Advisor:

Zilong Ye

Liaisons:

Andre Chen
Denny Ly
Karina Martinez
Vivian Sau

Table Of Contents

1. Introduction

1.1 Background:

Over the last 70 years, satellites have become a cornerstone of various processes. They make many conveniences of modern living possible through communication, GPS, weather prediction, research and defense, and more. Satellites have found their place as a common and important piece of modern-day technology. One important aspect of satellites that needs to be present in order for any of this to be possible, is the satellite's ability to pass on accurate data that has been gathered from the satellite to the ground system via uplinks and downlinks. Sometimes, the communication between the ground system and the satellite can have anomalies present. When this happens, it is critical that these anomalies are identified and corrected as soon as possible before they cause any serious consequences. The anomalies can result from an error in the hardware or software, outside environmental factors, or malicious attacks from hackers. Whatever the case may be, effective anomaly detection techniques should be present in order to identify any problems and ensure the long-term operation of a satellite in order to improve mission success. Therefore, in order to test effective anomaly detection techniques, California State University, Los Angeles has teamed up with the Aerospace Corporation, a leader in satellite technology, to use simulation tools to create a satellite simulation environment, simulate any anomalies that can arise, and test the different ways anomalies can be detected and corrected. The main tool that will be used throughout the Satellite Anomaly Injection and Detection Testbed (SAID) project is a platform called OpenSatKit (OSK).

Our project goal is to inject, detect and resolve two types of anomalies in OSK, Single Bit Error and Denial of Service.

- **Denial of Service**
    - Denial of Service occurs when communication systems are disrupted. Our injection of this anomaly utilizes a program to perform an SYN Flood which overwhelms the OSK software bus thus disrupting communication. When an abnormal amount of activity is detected, an anomaly alert is made. We use machine learning to predict and identify an anomalous attack in order to resolve it.
- **Single Bit Error**
    - Single Bit Error occurs when there is an error in data due to a flipped bit. Our injection involves using a command from the Memory Manager application to manually change a bit. Our detection software involves the use of the hamming code method to calculate the parity bits which would then be checked to see if

there was a flip or not. To detect a single bit error 2 commands need to be run from the Memory Manager application.

1.2 Design Principles:

The OSK applications for injection, detection, and resolution of each anomaly is the main deliverable and main design principle. We're mainly focusing on using the OSK framework to create applications that would help us in achieving detecting and resolving each anomaly. However, due to the limitations, we encountered when using the framework we could not implement a resolution to single-bit error and denial of service. This was another design principle of ours that was being limited by the number of ways we can implement our solutions for the detection and resolution of these anomalies. This design principle mainly came up because none of us were familiar with the OSK software before this year.

1.3 Design Benefits:

Utilizing OSK allows for a streamlined process. Instead of creating our own space and satellite simulation for the purposes of running our simulated anomalies, the tool is open-source and available for everyone to use and primed to make apps. Having access to last year's documentation allowed us an easier time getting an understanding of the process to create applications and commands for those applications. It also allowed us to get a better understanding of how to use OSK and all of its functions.

1.4 Achievements:

The achievements throughout the year include the creation of detection software for single-bit error and denial of service. In our detection software for single-bit errors, we used the hamming code method to keep track of the bits from a specific memory location. It would then store these bits in another memory location to check if there was a bit that was flipped. A second command would need to be called which is the one that does the comparison of the current bits to the previous bits that were stored. In our denial of service software, we use machine learning to predict and identify an anomalous attack. Although as a group we were not able to have a resolution to preventing SYN flood attacks, we did create a machine learning model that makes predictions on incoming packets, labeling them as either malicious or benign. We hope that next year's group can expand on this idea, and use our machine learning model as a filter allowing packets classified as benign to enter our server.

2. Related Technology

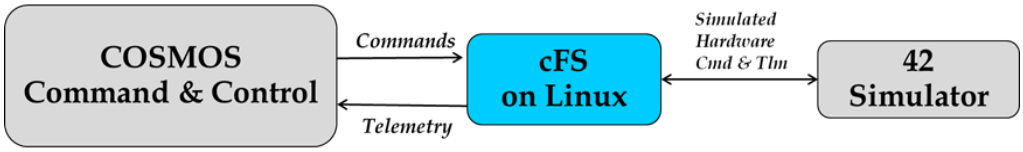
2.1 Linux:

The Linux operating system was used specifically for the Linux distribution of Ubuntu. This is where all the coding and testing was done to create the detection software for single-bit error and denial of service.

2.2 OSK:

OSK, also known as OpenSatKit, combines three powerful source tools that are currently used in real missions today. These include Ball Aerospace Corporation's COSMOS ground system, NASA Goddard's core Flight System(cFS) flight software, and NASA Goddards' 42 satellite simulator. Our goal with OSK is to use this simulation environment to simulate attacks against satellites and satellite malfunctions to produce effective countermeasures and remedies.

The 3 main components of OpenSatKit:



2.3 COSMOS - Ground System:

COSMOS is a suite of applications that can be used to communicate with the satellite, monitor its performance and health, and display its data. These systems can be anything from test equipment (power supplies, UPS devices, switched power strips, etc), development boards (Arduinos, Raspberry Pi, etc), to satellites. 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.

2.4 cFS - Flight Software:

OSK provides a complete desktop solution for learning how to use NASA's open-source flight software platform called the core Flight System(cFS). The cFS is a reusable flight software architecture that provides a portable and extendable platform with a product line deployment

model. The cFS has a significant flight heritage, prived a complete set of command and data handling functions required by most spacecraft, and is very reliable.

2.5 42 - Spacecraft Simulator:

OSK uses NASA Goddard's 42 dynamic satellite simulator for simulated hardware command and telemetry. The primary purpose of 42 is to support the design and validation of attitude control systems. 42 accurately models multi-body spacecraft attitude dynamics as well as modeling environments from low Earth orbit to throughout the solar system. It also features visualization of spacecraft attitude.

2.7 Existing Solutions:

Our team looked into many solutions for resolving and detecting single-bit errors. There were different solutions we saw when researching the one that was decided that seemed to work well for us involved the use of the hamming code method. This method allows us to mainly focus on a small number of bits instead of focusing on the entire string of bits. This method also allows us to easily expand on the number of bits we're looking at without slowing down the process of detecting single-bit errors.

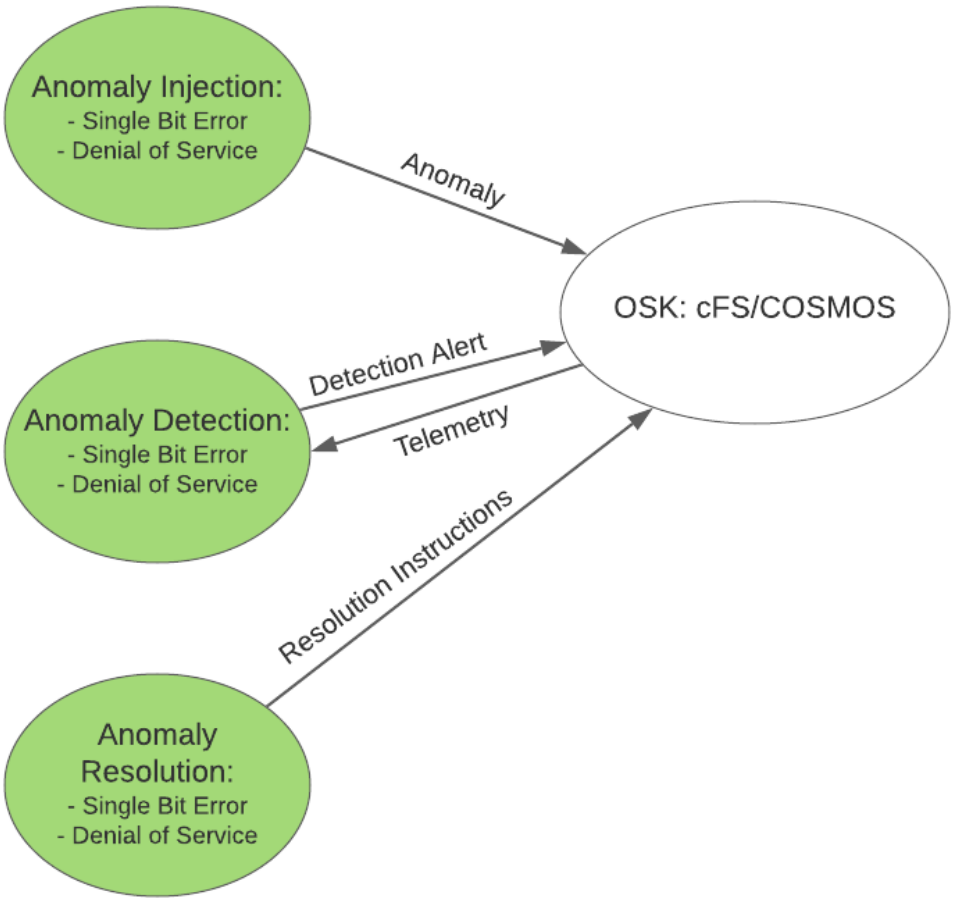
Our team also did research into how to resolve and detect denial of service. During our research, we could not find a resolution to denial of service that would work with the framework we were using. Solutions for detecting it mainly involve monitoring the network traffic which was done last year but this year we were going to be using machine learning to help detect it.

2.7 Reused Products:

Software that was created by the previous group that worked on this project is simulating a bit in memory flipping. The group was able to create the software that will flip a bit at a specific memory location and it will choose a bit at random from 8 bits. After the flipping has occurred it will also write to the CFS terminal letting the user know which bit was flipped and how the data looks after the flip. The reading and flipping of the bits use the core flight executive (CFE) API to read and write the bits from memory. Another core software that was created by the previous group and was used for this project involves the injection of denial of service. The injection involves a command that was created that utilizes a third-party software known as Netwox 76. This software simulates a syn flood attack to the hosting operating system.

3. System Architecture

3.1 System Architecture:



3.2 Software Development and Implementation:

3.2.1 Anomaly Detection

3.2.1.1 Single Bit Error: The detection software utilizes the core flight executes (CFE) API and the hamming code method. To detect single-bit errors the first command uses the CFE API to read the bits from an application and calculates the parity bits using the hamming code method to then store those parity in a dummy application. A second command then needs to be run which does the comparison of the current parity bits to the parity bits that were previously stored in the dummy application. It will then notify the user whether a flip has occurred or not. The comparison is able to be done because the CFE API allows us access to read and write the bits in memory.

3.2.1.2 Denial of Service: The detection software utilizes the KNeighbors Classifier, a machine learning algorithm, to determine whether there currently is a denial of service attack or not. When the command is run it will constantly be checking to see if there is an attack or not and when it determines there's an attack it notifies the user through the CFS terminal. It determines whether an attack is occurring or not by looking at the incoming packets and labeling them as benign or malicious.

3.2.3 Anomaly Resolution

3.2.2.1 Single Bit Error: Our group wasn't able to implement a resolution for single-bit error in OSK due to the limitations of OSK but we do have an idea on how we would implement it. The resolution would involve taking all the indices in the bit string where it's a 1 then converting them to binary and performing the XOR function on all of them. If there was no bit flip the result should then be 0 however, if a bit was flipped then the result would be the index where the bit was flipped but in binary. All that would need to be done is flipping the bit in that location and the single-bit error will be resolved.

4. Conclusion

4.1 Results

We were able to improve upon the previous detection software for detecting denial of service as well as creating detection software for single-bit error. Detection for single-bit error is able to display warning text when the detection command is run and it notices a change in the bits for a specific part in the memory. There are two commands for the detection software, the first command stores the parity bits of a specific memory location to another application's data and the second command then compares the parity bits of the original location and the second location to see if there is a difference. Detection denial of service software was developed. There is a command for the detection of denial of service that receives the data and sends it to an algorithm to determine if the data is a DDOS attack or not. The command would then create a small GUI that notifies the user an attack was occurring.

4.2 Future Additions

Future additions and improvements can be done to our project since we left a good start for future groups. With the use of our documents, it should be easy for future groups to understand the code that was done for the applications we used and created.

- A future addition that can be added to make the detection software better for the single-bit error would be to make it automatically catch when a bit is flipped instead of having to click on the commands. Even just combining the 2 commands that were created into 1 would be a huge improvement

- Another improvement would be to increase the number of locations where we are looking to see if there was a bit flipped as right now we're only looking at a specific memory location. Through the use of hamming code, it should be easy to look at even more bits as it won't take that much more memory to store the parity bits. The only way these methods can be added is if we're able to figure out how to read the data from two different applications as well as write to two different applications through one command as this was the main issue holding us back from making the detection better.
- For single-bit error, implementing our idea for the resolution of the single-bit error to OSK would be a good future feature as well.
- For denial of service, the machine learning algorithm used for the detection of denial of service can be improved with more representative data that will be used for training the model. The current dataset that was used for training was a rather smaller dataset.
- Another future addition for the detection of denial of service can be logging and tracking IP data for clients to analyze and monitor for potential attacks.

5. References

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

Previous group's work: https://github.com/PBJT2/CS4961