SYSTEMS ENGINEERING
Research Center

Security Engineering Project

Part 1a – System Aware Cyber Security for an Autonomous Surveillance System On Board an Unmanned Arial Vehicle

Part 1b – System Aware Cyber-Security Application to Unmanned Aircraft Systems (UAS) – Georgia Institute of Technology

Part 2 – Human Factors Engineering and System-Aware Cybersecurity

Part 3 – Cloud Architectural Assurance and Protecting Systems with Cloud-based System-Aware Methods

Part 4 – System-Aware, Model-based Cyber Assessments

Principal Investigator: Dr. Barry Horowitz, University of Virginia

Co PI:

Dr. Peter Beling, Associate Professor, University of Virginia

Dr. Kevin Skadron, University of Virginia

Dr. Ron D. Williams, University of Virginia

Dr. William Melvin, Georgia Tech Research Institute

Period of Performance: March 4, 2014 to January 31, 2015

Contract Number: HQ0034-13-D-004 Task order 0015, RT 115

Report No. SERC-2015-TR-036-4

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **31 JAN 2015**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **Security Engineering Project**	5a. CONTRACT NUMBER **HQ0034-13-D-0004**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Dr. Barry Horowitz Dr. Peter Beling Dr. Kevin Skadron Dr. Ron D. Williams Dr. William Melvin**	5d. PROJECT NUMBER **RT 115**
	5e. TASK NUMBER **TO 015**
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia Georgia Tech Research Institute**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Deputy Assistant Secretary for Defense for Systems Engineering, 4800 Mark Center Drive, 17C08, Alexandria, VA**	10. SPONSOR/MONITOR'S ACRONYM(S) **DASD (SE)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
Cyber Security, System Architecture, SE Security, The original document contains color images.

14. ABSTRACT
The Systems Engineering Research Center (SERC) has developed a novel cybersecurity concept for embedding security solutions into systems called System-Aware Cybersecurity. The overall goal of the System-Aware program is to develop low cost methods of protection against cyber exploits by our adversaries. Development of a prototype security system for securely monitoring an autonomous surveillance system on board an unmanned aerial vehicle for possible cyber attacks using reconfiguring systems for Sentinel-based architecture. Exploring decision support methodologies for determining on a mission basis the most critical system functions to secure employing attack tree tools and SysML/UML tools. Developing cyber sec CONOPS for operation of UAVâs that are possibly under attack. Exploring the opportunity to apply private Cloud capabilities as a Sentinel for monitoring ground-based systems so as be able to readily employ moving target and diversity solutions to secure the Sentinel and to monitor Cloud performance as a means for detecting possible cyber attacks. The major deliverables of the project were prototype-based and simulation experiments that result in enhanced understanding of requirements for cyber defense of systems and design patterns that can be reused across multiple system types

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a REPORT **unclassified**	b ABSTRACT **unclassified**	c THIS PAGE **unclassified**	**UU**	**269**	

This report is broken down into four major parts that correspond to the 4 major components of the research efforts for the RT-115 efforts for FY 2014. Each of the reports are standalone descriptions of the activities, the findings and the proposed future work for each of the focus areas.

Part 1a - System Aware Cyber Security for an Autonomous Surveillance System On Board an Unmanned Arial Vehicle – Describes the Phase 2 efforts to apply System-Aware techniques to protected an unmanned surveillance platform.

Part 1b - System Aware Cyber-Security Application to Unmanned Aircraft Systems (UAS) – Describes the Phase 2 efforts conducted by Georgia Institute of Technology.

Part 2 - Human Factors Engineering and System-Aware Cybersecurity – Begins addressing necessary human factors related solutions as part applying System-Aware Cyber Security solutions, addressing the necessity to instill confidence in needed operator decision-making under what can be very uncertain attack situations.

Part 3 - Cloud Architectural Assurance and Protecting Systems with Cloud-based System-Aware Methods – Addresses two efforts, 1) providing assurance techniques to private cloud architectures, and 2) providing protections to systems using System-Aware methods while taking advantage of the agility of private cloud platforms.

Part 4 – System-Aware, Model-based Cyber Assessments – Activities in architectural selection focused on investigating the hypothesis that a scalable and agile approach to the mission-focused architectural selection problem can be found by making use of, and integrating, two structured modeling approaches: System Models and Attack Models.

Security Engineering Project

Part 1a – System Aware Cyber Security for an Autonomous Surveillance System On Board an Unmanned Arial Vehicle

Principal Investigator: Dr. Barry Horowitz, University of Virginia

Co PI:

Dr. Peter Beling, Associate Professor, University of Virginia

Dr. Kevin Skadron, University of Virginia

Dr. William Melvin, Georgia Tech Research Institute

Technical Report SERC-2015-TR-036-4 Part 1a

January 31, 2015

Executive Summary

The Systems Engineering Research Center (SERC) has developed a novel cybersecurity concept for embedding security solutions into systems called *System-Aware Cybersecurity*. The overall goal of the System-Aware program is to develop low cost methods of protection against cyber exploits by our adversaries. Working through the SERC, the University of Virginia (UVa) and the Georgia Tech Research Institute (GTRI), Phase 1 efforts of the program were focused on advancing the System-Aware Cybersecurity concepts and evaluating a number of specific security design patterns that were intended to be reusable across a variety of applications. The major goal Phase 2 goal of the Sentinel program was to demonstrate the feasibility of System-Aware Cybersecurity design patterns designed in Phase 1 and to demonstrate the capabilities of a physical version of those protections to protect a system in a live environment and to experiment with the protections in order to monitor selected critical system functions of that system – in this case, an unmanned aerial vehicle (UAV). We call the physical implementation of this novel protection the Sentinel. Furthermore, those critical system functions were identified and analyzed for system vulnerabilities using an architectural selection methodology developed in Phase I of the project. Another goal of the Phase 2 project effort was to enhance and to progress that selection methodology in order to investigate more automated approaches to finding critical system functions in systems that can benefit from System-Aware protections in the face of potential threats emanating from such places as trusted insiders or compromised supply chains.

To demonstrate the effectiveness of the System-Aware design patterns in Phase 1, specific examples were developed for an unmanned aerial vehicle (UAV) application. In Phase 2 of the project, we developed and successfully tested an evaluation version of the System-Aware Cybersecurity protections (the Sentinel) designed in Phase 1. Designs were implemented to test the operational feasibility for the protections of various navigation, command and control, and payload hardware and software systems onboard an automated surveillance platform and in the supporting operator ground-station. Our chosen test UAV case was the GTRI Aerial Unmanned Sensor System (GAUSS) aircraft. The GAUSS platform is a small research UAV with a widely used, commercial off-the-shelf autopilot system and camera gimbal. The demonstration showed how the System-Aware approach can be used to thwart cyber-attacks against autopilot and sensor systems. Live flight experimental testing was accomplished during flight tests, which were run in Blakely, GA during the week of October 7th-10th, 2014.

During the Phase II effort the UVa/GTRI team achieved a number of significant accomplishments during the flight evaluation:

- Miniaturization of the System-Aware protections to fit the power usage and form-factor requirements of our test UAV.
- Simulated attacks were developed for a sampling of typical UAV systems including:
  - The navigation systems including a GPS walk-off attack and attacks on the flight plan waypoint controls and attacks
  - The flight payload systems including the camera gimbal controls and on the video imagery exploitation system.
- Built a prototype smart security Sentinel to host System-Aware Cybersecurity solutions to protect against the sample cyber-attacks.
- Successfully demonstrated the Sentinel protections against attacks in live-flight evaluations.
  - Created a data set including video, screen recordings system data and Sentinel related data for all flight experiments.
- Integrated the new SiCore technology designed in Phase 1 into the protections for the Sentinel platform.

Table of Contents

1 Project Status Overview 11

2 Project Emulation and Simulation and Flight Evaluation Environments 12

2.1 UVa Piccolo II HiL Emulation Environment 12

2.2 GTRI HiL Emulation Environment 13

2.3 Phase 2 Flight-Ready Integrated System Environment 16

2.3.1 Flight-test Hardware in the Loop Simulator Testing 16

3 Review of Phase II Activities 19

3.1 System-Aware Architectural Selection and Assessment Methodology 19

3.1.1 Architectural Selection and Assessment 20

3.1.1.1 Definitions 20

3.1.1.2 Methodology Process Steps 20

3.1.1.3 Vulnerability and Threat Analysis Process 22

3.1.1.4 System Functions and their Interrelations 22

3.1.1.5 Identifying and Classifying Potential Cyber Attacks 26

3.1.1.6 Selection of Cyber Attacks to move forward 28

3.1.1.7 Understanding Attack Surfaces 29

3.1.2 Cybersecurity Architectural Selection – a UAV Case Study 29

3.1.2.1 Selection of System Functions for Protection 32

3.1.2.2 Identification of Asymmetric Attack Vectors 32

3.1.2.3 Selection of System-Aware Design Patterns 32

3.1.2.4 Determining the Security Effectiveness 33

3.1.2.5 Determining the Resources Necessary to Develop the System-Aware Architecture 33

3.1.2.6 Selecting Architectural Candidates 34

3.1.2.7 Limitations of the Proposed Enhanced Architectural Scoring Framework 35

3.1.3 Prototype Architectural Assessment Support Tool 35

3.1.4 Selection of System Functions for Protection 35

3.1.4.1 Selection of System-Aware Design Patterns 36

3.1.4.2 Determining the Resources Necessary to Develop the System-Aware Architecture 37

3.1.4.3 Determining the Security Effectiveness 38

3.1.4.4 Selecting Architectural Candidates 39

3.1.4.5 User Constraints 39

3.1.4.6 Automated Support to Generate Candidate Architectures 40

3.1.4.7 Tools to Support User Exploration 41

3.1.5 Additional Use Case – The Navy 10th Fleet 46

3.2 Attack Development 48

3.2.1 Parameter-Based System Attack 48

3.2.1.1 Parameter-Based Attack Implementation Details 49

3.2.1.2 Masking the Parameter-Based Attack to Prevent User Detection 50

3.2.1.3 Implementation of the Parameter-Based Attack by Compromising the PCC 51

3.2.1.4 Flight Demonstration - The Waypoint Attack 52

3.2.2 GPS System Attacks 55

3.2.2.1 Applying the Relational System Aware Architectural Assessment Methodology to GPS55

3.2.2.2 Introduction to GPS System Attack 62

3.2.2.3 High Level Description of GPS Attack 62

3.2.2.4 GPS System Attack Specifics 63

3.2.2.5 The Attack Simulation 63

3.2.2.6 Flight Demonstration - GPS Walk-Off Attack 65

3.2.3 Gimbal System Control Attacks 66

3.2.3.1 Introduction to Gimbal Attacks 66

3.2.3.2 High Level Description of Gimbal Attack 67

3.2.3.3 Gimbal Attack Specifics 67

3.2.3.4 Flight Demonstration - Gimbal Command Attack 68

3.2.4 Hardware Security Against Design and Manufacturing Attacks 69

3.2.4.1 Introduction to Design and Manufacturing Attacks 69

3.2.4.2 Flight Demonstration - High Level Attack Scenarios 70

3.3 Design and Development of the Super Secure, Sentinel 70

3.3.1 Sentinel Platform Development 70

3.3.1.1 Single Board Computers 70

3.3.1.2 SiCore SHIELD 2 Coprocessor 72

3.3.2 Attack Detection, Mitigation and Restoration 73

3.3.2.1 Parameter-Based Attack Detection, Mitigation, and Restoration 73

3.3.2.2 GPS System Attack Detection and Mitigation 75

3.3.2.2.1 Frame of Discernment........ 76

3.3.2.2.2 Similarity Measurement........ 76

3.3.2.2.3 Analytical Equivalent Pairings 77

3.3.2.2.4 Sequential Change Detection........ 79

3.3.2.2.5 Flight Demonstration - Navigation System Protection 83

3.3.2.2.6 Issues We Addressed in the GPS Development Efforts 84

3.3.2.3 Gimbal Attack Detection and Mitigation........ 88

3.3.2.3.1 Using Mission Context to Detect Gimbal Attacks 88

3.3.2.4 Packet Detection 89

3.3.2.5 Retract/Deploy Command Detection........ 89

3.3.2.6 Erratic Gimbal Command Detection 89

3.3.3 Hardware Security Against Design and Manufacturing Attacks – Securing the Protocol Conversion 90

3.3.3.1 An FPGA-Based Application of Protocol Conversion with TMR 90

3.3.3.2 Protocol Conversion 91

3.3.3.3 SD Card Access 91

3.3.3.4 Design Implementation........ 91

Further Modifications to SiCore SHIELD card for the Sentinel creating the SHIELD 2 card........ 91

3.3.3.5 Protocol Conversion 93

3.3.3.6 SD Card Access 94

3.3.3.7 TMR – Triple Modular Redundancy........ 95

3.3.3.8 Lessons Learned 97

3.3.3.8.1 Generating VHDL PROM Module from C Code 97

3.3.3.8.2 Sending Packets in uIP........ 97

3.3.3.8.3 TCP Delayed ACK 98

3.3.3.8.4 Bug in uIP 98

3.3.3.8.5 Bug in ETH driver 99

3.3.3.8.6 UART Flow Control Signals 99

3.3.3.8.7 Clock Signal for Ethernet PHY 99

3.3.3.8.8 Pin Assignments for LDs 99

3.3.3.8.9 Buffer Size of UART Transceiver 100

3.3.3.8.10 Timing Requirements 100

3.3.3.8.11 Synchronization 100

3.3.3.8.12 Solutions and Detection/Restoration Mechanisms 103

3.4 Evaluative Criteria 103

4 Phase II Flight Evaluation 105

4.1 Flight Evaluation Results Overview 105

4.2 Development of the Serial Spy 109

4.3 Development of the Tester’s Interface 111

4.4 Aircraft Integration 119

4.5 Flight Test Demonstration Attacks 120

4.6 Sentinel Protections For Selected Attacks 120

4.6.1 Gimbal Attack Detection and Mitigation 120

4.6.1.1 Using Mission Context to Detect Gimbal Attacks 121

4.6.1.2 Packet Detection 121

4.6.1.3 Retract/Deploy Command Detection 121

4.6.1.4 Erratic Gimbal Command Detection 122

4.7 Cyber Commander Interface Design and Development 123

4.7.1 Development of Interface 124

4.7.2 Visibility of system status 124

4.7.3 Match between system and the real world 125

4.7.4 Flexibility and efficiency of use 125

4.7.5 Aesthetic and minimalist design 125

4.7.6 Recognition rather than recall 125

4.7.7 RabbitMQ 126

4.7.8 GPS Parameter Display 127

5 Conclusions and Future Work 128
6 Appendix 130
6.1 Supporting Calculations for GPS System Attack 130

List of Figures

Figure 1. Basic HiL configuration for the Piccolo II. 13
Figure 2. GTRI HIL Simulator with System Aware Cyber Security Components Installed 14
Figure 3. Data flow diagram for the GTRI HiL emulation environment. 15
Figure 4. ViewPoint user interface for streaming video created by the MetaVR scene generator 15
Figure 5. Airborne System Architecture 17
Figure 6. Ground System Architecture 18
Figure 7. UAV Onboard Systems showing the four major system groups. 24
Figure 8. Influence diagram for used to understand the relationship between the UAV subsystems used for navigation. 25
Figure 9. Influence diagram used to understand the relationship between the UAV subsystems used for gathering surveillance data. 26
Figure 10. Cyber-taxonomy. 28
Figure 11. Attack Surface Concept Model. 29
Figure 12. Flowchart illustrating the Cybersecurity Architectural Selection framework.. 31
Figure 13. The prototype decision support system's support for selecting functions to protect 36
Figure 14. Illustrates how the prototype system is used to support the selection of System-Aware design patterns. 37
Figure 15. Illustrates how the prototype system supports the assessment of the necessary resources needed to implement a given System-Aware architecture.. 38
Figure 16. Illustrates how the prototype supports the assigning of security effectiveness scores to all combinations of system function and design pattern. 39
Figure 17. Illustration of the budget slider used in the prototype system. 40
Figure 18. The architecture candidate created automatically using the *Blue Perspective* 43
Figure 19. The architecture candidate created automatically using the *Red Perspective* 43
Figure 20. Quad chart displaying all of the selected combinations of system functions and design patterns in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. 44
Figure 21. Quad chart displaying all of the combinations of system functions and design patterns not included in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. 44
Figure 22. Set of tools the user can use to adjust the candidate architectures generated by the prototype decision support system (*Blue Perspective* and the *Red Perspective*). T 45
Figure 23. Illustrates how the prototype decision support system supports the user as they adjust the candidate architectures into new architectures. 45
Figure 24 - Embedded attack platform for masking the operator display on a SBC. 51
Figure 25. Piccolo II HiL configuration with Embedded Attack Platform 52
Figure 26. Waypoint Attack Flight Plans 54
Figure 27. Camera gimbal influence diagram. 55
Figure 28. Attack tree for GPS attacks to alter georeference data. 56
Figure 29. Pruned Attack Tree. 58
Figure 30. Autopilot to Simulator communications. 63
Figure 31. Simulated attack configuration. 64
Figure 32. Example of GPS attack. 65

Figure 33. GPS walk-off attack region shown as upper right orbit. Typical flight path shown as right hand pattern on Runway 23 66
Figure 34. Gimbal command attack region shown as upper right orbit. 69
Figure 35. Raspberry Pi SBC. 72
Figure 36. Super secure smart Sentinel for protection with a designated cyber security officer. 74
Figure 39. Architecture for camera system. 81
Figure 40 - SiCore SHIELD II Board 92
Figure 41 A Block Diagram of the LEON3 Architecture 93
Figure 42 Implementation of TCP Stack 94
Figure 44 States of Sending TCP Packets in uIP 98
Figure 45 Waveform of Synchronization Point 102
Figure 48. Simplified Block Diagram Showing SerialSpy Usage in a UAV Sentinel 109
Figure 49. FalconView Display on the Tester’s Interface 112
Figure 50. HUD Indicators 112
Figure 51. Tester’s Interface Showing Truth and Attack Positions for the Aircraft 113
Figure 52. Tester’s Interface Showing the Defense Position for the Aircraft (in green) 113
Figure 53. Tester’s Interface 114
Figure 54. UDP Connections Control (a) Sockets not Connected, (b) Sockets Connected 114
Figure 55. Defense Section of the Tester’s Interface 115
Figure 56. Attack Section of the Tester’s Interface 116
Figure 57. ViewPoint Plugin Section 117
Figure 58. Attack Pi Section 118
Figure 59. Masking Section of Tester’s Interface (Orbit Waypoint Selection) 118
Figure 60. Masking Section of Tester’s Interface (Waypoint List Selection) 119
Figure 61. Payload Chassis 120

List of Tables

Table 1. Waypoints on North West side of runway 53
Table 2. Waypoints centered over runway 54
Table 3. Impact of design patterns on the system. 60
Table 4. Effects of System-Aware defense on the system and attacker for the gimbal GPS metadata attack. 61
Table 5. Frame of Discernment for Navigation Architecture. 76
Table 6. FOD with Altimeter and Location Estimator. 78
Table 7. Navigation procedures. 79
Table 8. Procedures for five navigation components. 80
Table 9. FOD for camera system. 82
Table 10. Procedures for camera system. 83
Table 11 Different Pin Assignments on NetFPGA and SiCore Board 100
Table 12. Commands that the SerialSpy searched for in Gimbal Communications 110

1 Project Status Overview

The Systems Engineering Research Center (SERC) has been engaged with the Department of Defense (DoD) in developing a novel cyber security concept for embedding security solutions into systems; this new concept is referred to as *System-Aware Cyber Security*. These solutions provide greater assurance to the most critical system functions by providing an additional layer of defense that complements perimeter and network security solutions that serve to guard the entire system from penetration. System-Aware solutions are effective at guarding against insider and supply chain attacks in addition to attacks that circumvent perimeter security solutions. The broad objective of the System-Aware program can be thought of as reversing cyber security asymmetry from favoring our adversaries, to favoring the US; i.e., requiring adversaries to make large investments in developing complex cyber exploits based on low cost System-Aware cyber security solutions for protecting critical system functions.

To-date, the SERC and a University of Virginia (UVa) led team, consisting of the UVa and the Georgia Tech Research Institute (GTRI), have advanced the System-Aware cyber security concept and evaluated a number of specific design patterns that are intended to be reusable across a variety of applications. These patterns include, but are not limited to, employing diverse redundant components in critical subsystems, using voting techniques across diverse redundant components for real-time discovery and elimination of infected components, dynamically modifying the configuration of software components in systems through virtual configuration hopping techniques, dynamically modifying the configuration of the hardware/software components in systems through physical configuration hopping techniques, using system specific data consistency-checking to determine if critical system information has been manipulated, and where applicable, use of analog components as trusted elements to perform critical security functions in systems. Furthermore, a decision support framework has been developed for use by systems engineering teams in selecting a subset of available design patterns for integration into a cyber security system architecture. A central part of the System-Aware concept is to implement the Sentinel in a manner that is expandable and to include a high degree of advanced security features.

In addition, a Phase 0 effort consisting of an evaluation of possible applications of existing, developed design patterns as part of the SERC-sponsored RT-28 design efforts undertaken in FY 2013 to a specific application has been completed. The results of this effort identified an unmanned air vehicle (UAV) system configured for conducting surveillance missions as suitable for a follow on Phase 1 and Phase 2 prototyping pilot efforts for validating the System-Aware Cybersecurity concept. The application to UAV-based systems was inspired by the wide variety of subsystems that are used in UAV configurations, the range of potential cyber-attacks that can seriously impact the critical missions of these systems, and the significant power, space and performance constraints that System-Aware designs must address in order to operate in UAV-based configurations. The successful simulation prototypes in Phase 1 led to the design of a Phase 2 prototype that shows the feasible capabilities of using System-Aware techniques in a live-flight demonstration system that was successfully delivered and demonstrated in flight evaluations in Blakely, GA in October, 2014.

This document outlines the Phase 2 project that consisted of activities undertaken to integrate the results of the Phase 0 and 1 efforts into the GTRI Aerial Unmanned Sensor System (GAUSS) aircraft in order to create and to evaluate a flight-ready demonstrations of the System-Aware protections. In Phase 2, we accomplished the following:

- Implementation of the System-Aware protections to fit the power usage and form-factor requirements of our test UAV.
- Development of attacks and protections for a sampling of typical UAV-based systems including attacks on:
    - The navigation systems including a GPS walk-off attack and attacks on the flight plan waypoint controls.
    - The flight payload systems including the camera gimbal controls and on the video metadata that supports ground-based imagery exploitation.
- Development of a prototype, smart security Sentinel to host System-Aware Cybersecurity solutions and to protect against the sample cyber-attacks in live flight tests on various sub-systems.
- Successful demonstration of the Sentinel protections against attacks in live-flight evaluations in Blakely, GA during October 2014.
    - Creation of a data set from experiments including video, screen recordings of system data and Sentinel related data for all flight experiments.
    - Creation of a video in support of efforts to raise awareness of the System-Aware Cybersecurity techniques to organizations outside DoD.
- Integration of the new SiCore implementation technology designed in Phase I into the protections for the Sentinel platform itself.

2 Project Emulation and Simulation and Flight Evaluation Environments

The platform selected for demonstrating methodologies to protect unmanned autonomous systems (UASs) from cyber-attacks is the GTRI GAUSS aircraft. This UAV uses the Piccolo II unmanned aerial avionics system (hereafter referred to as Piccolo or Piccolo II) and a TASE 150 camera gimbal system, both supplied by Cloud Cap Technology™, a United Technology Corporation™ company. Prior to flight-testing any new technology with this aircraft, extensive testing is conducted using ground-based simulators and emulators to ensure flight safety. The Piccolo II autopilot system supports both a software-in-the loop (SiL) simulation capability and a hardware-in-the-loop (HiL) emulation capability. These simulation environments have been used in Phase 1 and 2 of the project for testing and evaluation of the System-Aware protections to be highlighted in our flight evaluations. Both the UVa and the GTRI have versions of the SiL and HiL environments for supporting on-site development and testing at their respective locations. The GTRI emulator also includes the capability to integrate the TASE camera gimbal into HiL emulation environment. Following the tests conducted in the simulation environments, there is extensive testing that is accomplished on the aircraft platform itself before certification for live flight tests. The following subsections describe the SiL and HiL development and test environment at each project location as well as the flight testing environments and flight configurations used during flight evaluations that were conducted in October 2014 at the Early County Airport in Blakely, Georgia.

2.1 UVa Piccolo II HiL Emulation Environment

The UVa utilizes the out-of-the-box simulation and emulation capabilities provided by Cloud Cap Technology with their Piccolo II autopilot. The HiL emulation environment uses a simulator to represent the state of the aircraft (e.g., the power levels, aileron settings, and fuel), as well as to generate GPS data to simulate the aircraft flying over any location. The actual control of the aircraft is accomplished by using the Piccolo II operator interface for remote operator control of the Piccolo II, and the supporting ground transmitters and receivers. As seen in Figure 1, the operator interface, Piccolo Command Center (PCC), connects directly to a ground station. This ground station is used to send and receive commands and status information from the Piccolo II. For the HiL emulation environment, the Piccolo II is connected to a PC that hosts a simulator of the aircraft as well as of the GPS satellites. This computer can be the same one that is hosting the PCC or a separate computer, as depicted in Figure 1.

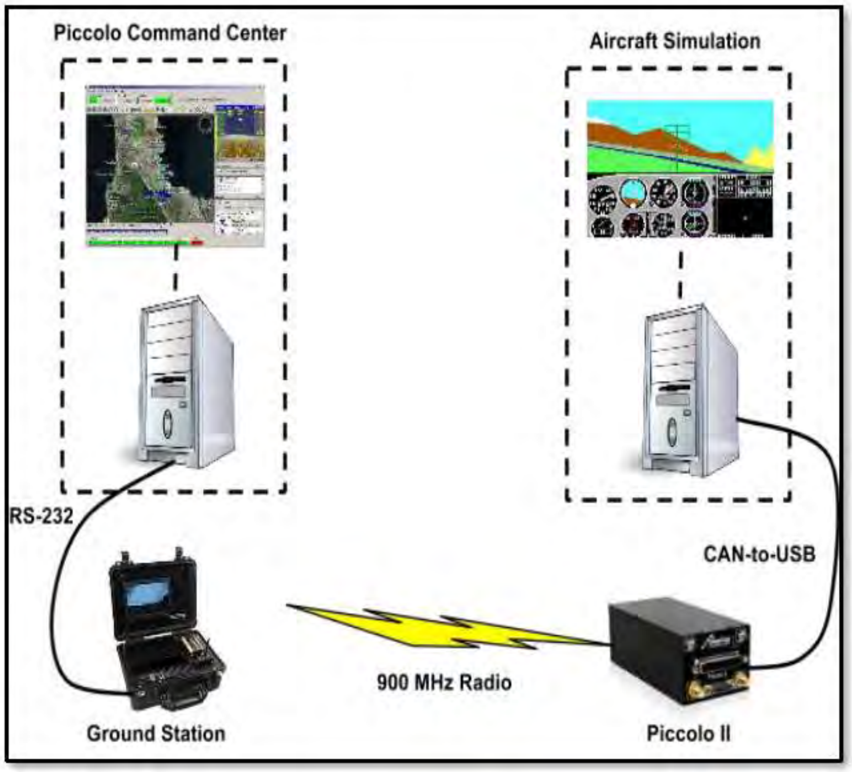


Figure 1. Basic HiL configuration for the Piccolo II.

In both the UVa and the GTRI HiL emulation environments, the waypoints for the flight path are sent from the operator's interface (PCC) to the Piccolo II via a radio link between the ground station and the Piccolo II; this is the same link that is used in an actual flight. A six degree of freedom (6 DoF) flight dynamics model of the aircraft running on the aircraft simulator computer provides the aircraft's state to the Piccolo II via a CAN bus (controller area network). The Piccolo II calculates the aircraft's actuator commands and sends them to the 6 DoF simulation via the same CAN bus. In the GTRI HiL, the aircraft's pose (position and attitude) are also sent to the gimbal via the CAN bus to simulate the output of its own integrated GPS and inertial measurement unit.

The UVa team has leveraged this emulation capability as it has designed the system attacks, the Sentinel monitoring and detections capabilities, and any restorative actions using this environment. Phase 2 tests were conducted in the HiL simulation before the flight evaluations in order to verify system functions before being shipped to GTRI for their testing in their simulation and on the actual flight hardware platform. The details of how the HiL emulation environment was augmented to include the Sentinel's monitoring, detection, and restoration techniques.

2.2 GTRI HiL Emulation Environment

The GTRI team utilized a Piccolo HiL emulation/simulation environment that is identical to the UVa environment with the addition of the TASE 150 gimbal system from Cloud Cap Technology and its associated hardware and software. As seen in Figure 2, the GTRI's HiL emulator consists of a Piccolo II autopilot, TASE 150 gimbal, Cloud Cap video processing system (VPS), and ground station.

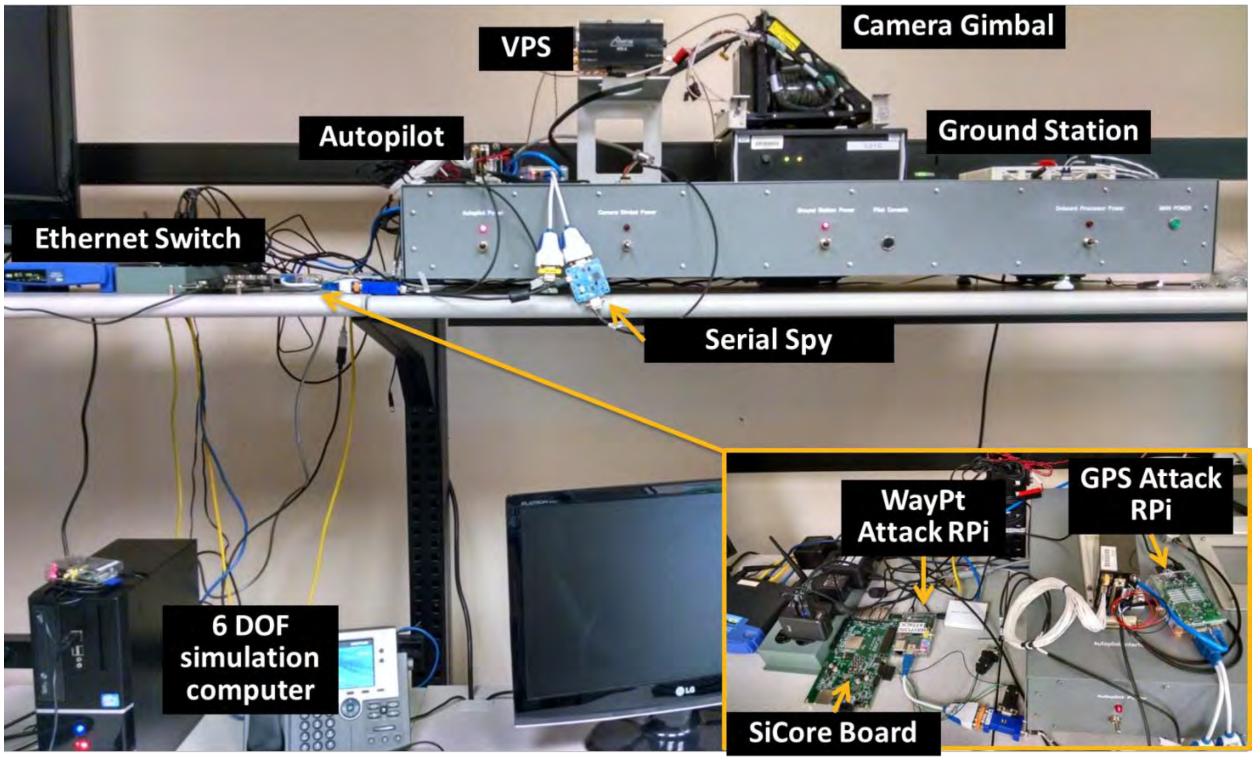


Figure 2. GTRI HIL Simulator with System Aware Cyber Security Components Installed

Figure 3 shows the data communications between the various avionics systems for the HiL environment shown in Figure 2. The aircraft pose data is used by the gimbal system to automatically steer the gimbal when it is locked on a point of interest (POI). The gimbal outputs metadata to the VPS which overlays gimbal status information on the video via a serial line using the RS-232 standard; e.g., the current pan, tilt, and zoom of the camera. The camera also sends NTSC analog video to the VPS which performs image stabilization before sending the video to the video display via a radio link. In the HiL emulation environment, the analog video from the camera is replaced with synthetic video from a scene generator, MetaVR. MetaVR, depicted in Figure 4, is an additional HiL capability that allows for the visualization of camera imagery of an aircraft in flight. MetaVR, a virtual reality scene generator, decodes the state information of the aircraft via a network interface with ViewPoint, the program used to view the video. The software generates a scene based on the aircraft GPS information and gimbal angles. Any scene within the Southeast United States or Afghanistan can currently be generated, allowing a variety of CONOPS to be visualized. The analog video is converted to a digital format (H.264) and displayed at the video display using the ViewPoint software.

Figure 3 also shows the location of data monitoring points on the serial connections that are used to detect the injection of a cyber-attack. To conduct this monitoring GTRI developed a snooping device known as a Serial Spy that is a microcontroller-based device designed to monitor and/or manipulate RS-232 serial data between two devices. (see section 4.2). These snoopers can intercept the serial data, decode the information, and retransmit the data into the system.

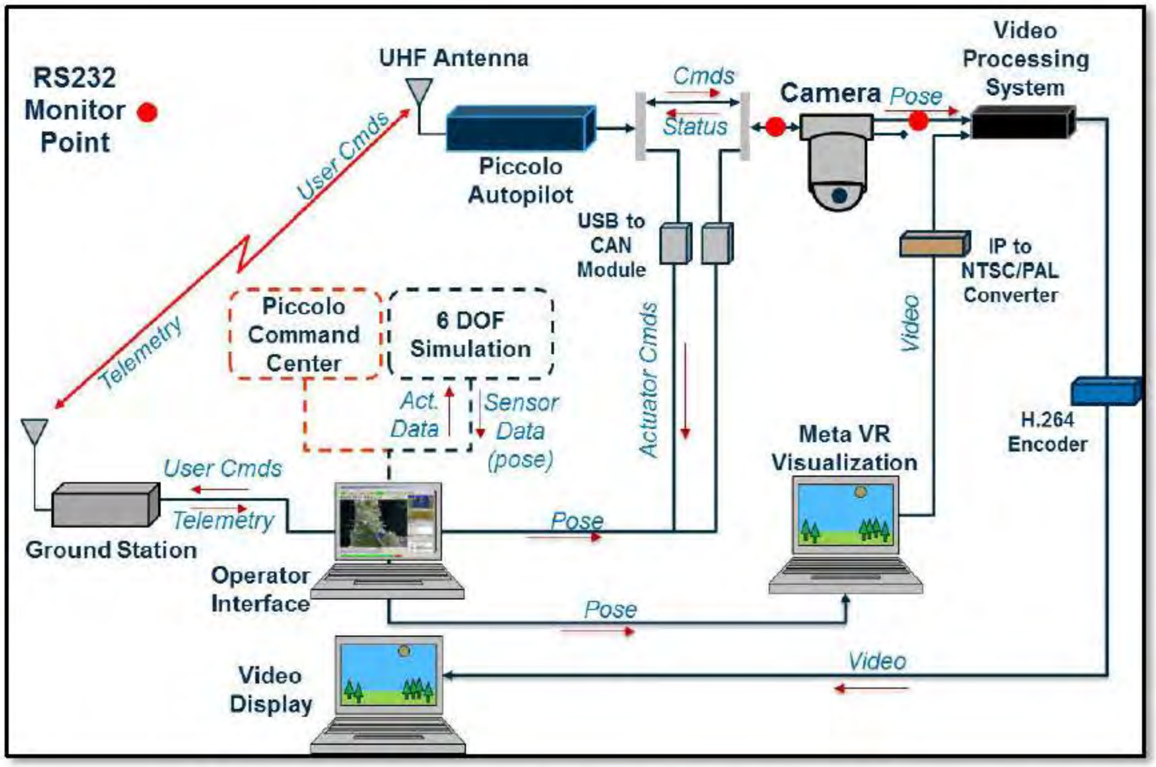
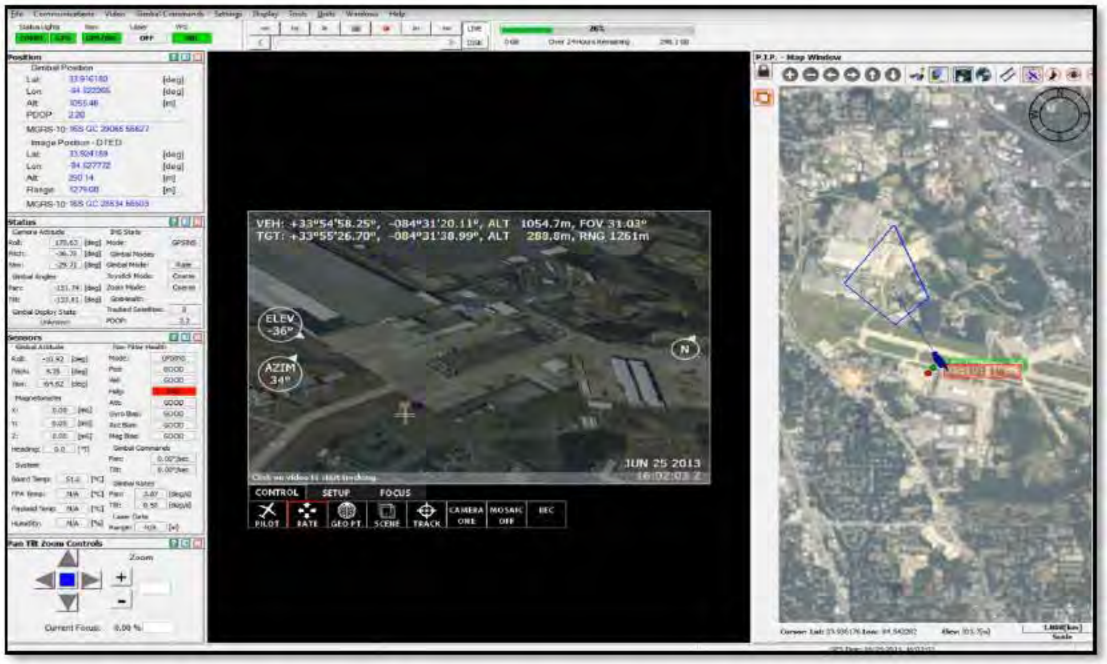


Figure 3. Data flow diagram for the GTRI HiL emulation environment.



VEH: +33°54'58.25", -084°31'20.11", ALT 1054.7m, FOV 31.03°
TGT: +33°55'26.70", -084°31'38.99", ALT 288.8m, RNG 1261m

JUN 25 2013
16:02:03 Z

Position
Gimbal Position
Lat: 33.916180 [deg]
Lon: -84.522265 [deg]
Alt: 1055.48 [m]
PDOP: 2.20
MGRS-10: 16S GC 29065 55827
Image Position - DTED
Lat: 33.924189 [deg]
Lon: -84.527772 [deg]
Alt: 290.14 [m]
Range: 1279.08 [m]
MGRS-10: 16S GC 28534 56503

Figure 4. ViewPoint user interface for streaming video created by the MetaVR scene generator

2.3 Phase 2 Flight-Ready Integrated System Environment

During Phase 2, technical designs developed during Phase 1 were implemented into a version that could fit into the power and size requirements of the GAUSS platform used for the flight evaluations on this project. The attacks we addressed during the evaluations required both air and ground components in order to provide System-Aware monitoring, detection, alerting and protection capabilities to the system operators and to protect against attacks that were coordinated on the aircraft and on the ground. Figure 5 shows the final configuration of the configurations used in the airborne portion of the protections during flight-testing. This figure highlights the components that were used to inject attacks, the hardware that housed the System-Aware technologies and the additional sensors that were added to the system (such as diverse, redundant GPS receivers) for use by the System-Aware monitoring capabilities. Figure 6 shows the corresponding ground architecture that was utilized during the flight evaluations. Attacks such as the Stuxnet attack have highlighted the need for protections against attacks where persistent effects on the systems under attack were hidden from the operators. Our ground and air System-Aware protection architectures provide a complete solution for monitoring system behaviors for potential threats. In this case, the coordinated attack we chose to highlight was a masking attack that hides an attack on flight-plan waypoints occurring on the airborne platform from the operators and their command and control systems on the ground.

2.3.1 Flight-test Hardware in the Loop Simulator Testing

Prior to any flight test activity, extensive testing of the Sentinel system was conducted in the GTRI hardware in the loop (HIL) simulator. The components integrated into the GTRI HIL simulator are indicated in Figure 5 and Figure 6 while Figure 2 shows the actual simulator. Three Raspberry Pi single board computers (shown in orange as Attack Pi 1 and Attack Pi 2 in Figure 5 and as Masking Attack Pi in Figure 6) are used to effect the cyber-attacks. Attack Pi 1 executes the waypoint attack onboard the aircraft that changes the list of waypoints in the autopilot's flight plan. This attack causes the aircraft to deviate from its intended course. The Masking Attack Pi at the ground control station hides the fact that the aircraft has gone off course. The second onboard attack Pi intercepts and corrupts the GPS position data being sent from the autopilot to the camera gimbal. This attack corrupts the target geo-location information that is included as metadata in the video stream. Also included in the attack system is an onboard Ethernet switch to allow developers and the test director to communicate with the attack devices.

The components shown in green in Figure 5 and Figure 6 constitute the cyber defense systems. The primary aircraft component is the SiCore Shield II computer that serves as the onboard Sentinel. A Raspberry Pi SBC (the Ground Sentinel Pi in Figure 6) works in conjunction with the onboard Sentinel to detect waypoint attacks even if they are masked. The Passive serial splitter is used to monitor the autopilot's flight plan through the autopilot's COM port 3 as part of the defense against waypoint attacks.

The SerialSpy shown in Figure 5 monitors the communications between the autopilot and the gimbal. This device defends against GPS attacks by checking the autopilot's GPS position data against validated position data provided by the Sentinel. A detailed description of the SerialSpy is provided in Section 4.2.

Three additional Raspberry Pis (labeled Air Sentinel 1, Air Sentinel 2, and AS3 in Figure 5) are included as part of a triple redundant voting scheme to validate GPS position data. Two of these Raspberry Pis have independent GPS receivers while the third uses the autopilot GPS data acquired via the Ethernet link. If the GPS position data does not match the validated data within some user-controllable error bound, an attack is indicated and the SerialSpy passes on to the gimbal the validated position data provided by the Sentinel.

The final component of the cyber defense system is the Cyber Commander Station (on the right in Figure 6). This is an application that runs on the ground control station and provides alerts to the operator in the event of an attack.

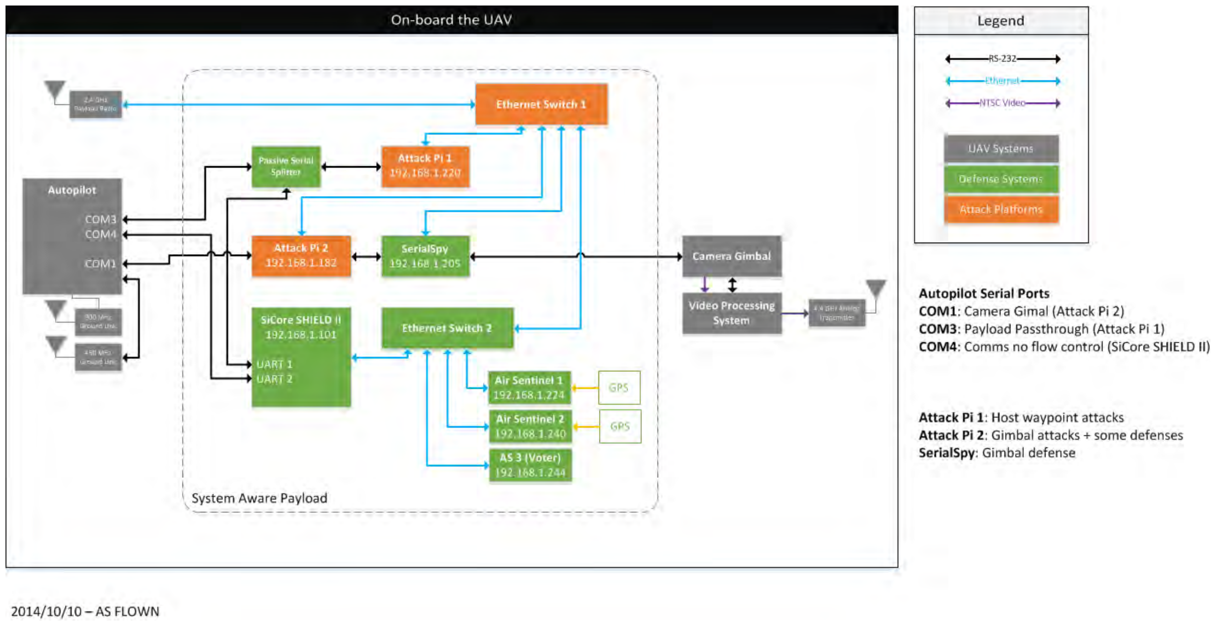

On-board the UAV
2.4 GHz Payload Radio
Ethernet Switch 1
Passive Serial Splitter
Attack Pi 1 192.168.1.220
Autopilot
COM3
COM4
COM1
Attack Pi 2 192.168.1.182
SerialSpy 192.168.1.205
Camera Gimbal
Video Processing System
2.4 GHz Analog Transmitter
900 MHz Ground Link
450 MHz Ground Link
SiCore SHIELD II 192.168.1.101
UART 1
UART 2
Ethernet Switch 2
Air Sentinel 1 192.168.1.224
Air Sentinel 2 192.168.1.240
AS 3 (Voter) 192.168.1.244
GPS
GPS
System Aware Payload
Legend
RS-232
Ethernet
NTSC Video
UAV Systems
Defense Systems
Attack Platforms
Autopilot Serial Ports
COM1: Camera Gimal (Attack Pi 2)
COM3: Payload Passthrough (Attack Pi 1)
COM4: Comms no flow control (SiCore SHIELD II)
Attack Pi 1: Host waypoint attacks
Attack Pi 2: Gimbal attacks + some defenses
SerialSpy: Gimbal defense
2014/10/10 – AS FLOWN

Figure 5. Airborne System Architecture

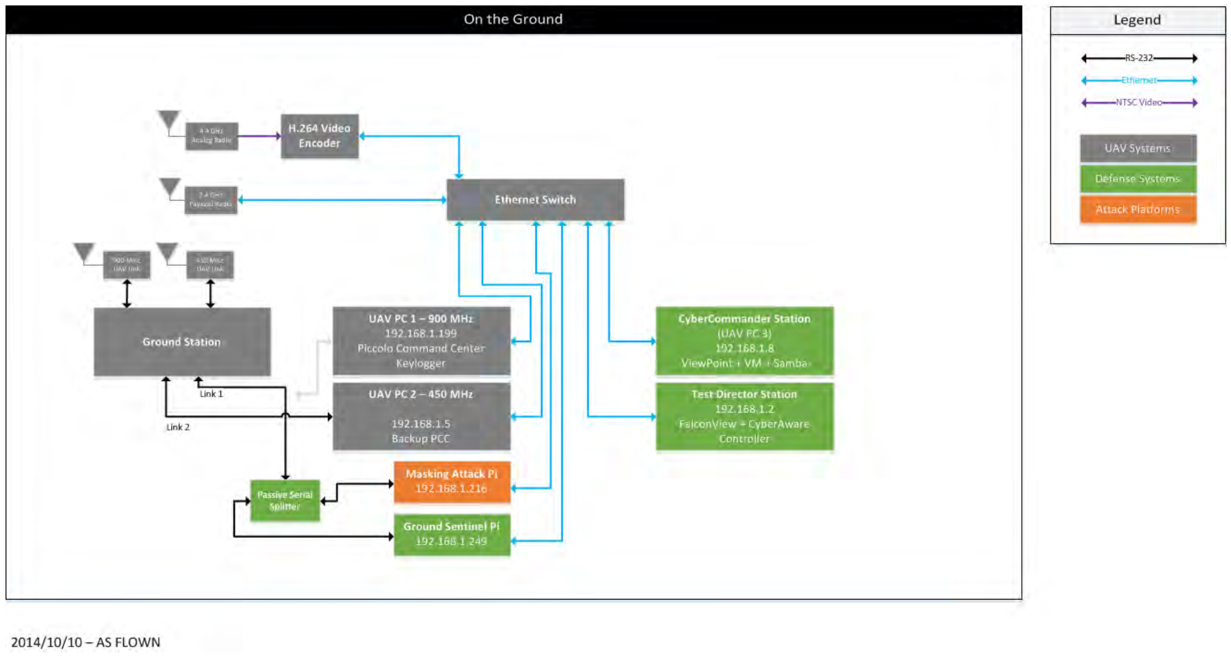

On the Ground
Legend
RS-232
Ethernet
NTSC Video
UAV Systems
Defense Systems
Attack Platforms
4.4 GHz Analog Radio
H.264 Video Encoder
2.4 GHz Payload Radio
Ethernet Switch
900 MHz UAV Link
450 MHz UAV Link
Ground Station
UAV PC 1 – 900 MHz 192.168.1.199 Piccolo Command Center Keylogger
UAV PC 2 – 450 MHz 192.168.1.5 Backup PCC
CyberCommander Station (UAV PC 3) 192.168.1.8 ViewPoint + VM + Samba
Test Director Station 192.168.1.2 FalconView + CyberAware Controller
Link 1
Link 2
Passive Serial Splitter
Masking Attack Pi 192.168.1.216
Ground Sentinel Pi 192.168.1.249
2014/10/10 – AS FLOWN

Figure 6. Ground System Architecture

Before designing and developing the attacks and associated protections for the UAV system, the UVA and GTRI teams used a new methodology to identify the critical system functions that were to be protected by the System-Aware technology. This method is known as the System-Aware Architectural and Assessment methodology and is described below.

3 Review of Phase II Activities

3.1 System-Aware Architectural Selection and Assessment Methodology

The System-Aware Architectural Selection and Assessment methodology is a process that has been developed as part Phase 1 and Phase 2 of this project in order to identify the critical system components for a particular system, to identify the possible attack paths to attack those components, to determine which of those attack paths would be most desirable an adversary, to identify possible cyber security defenses against those attacks as well as to evaluate the impacts of those defenses on the attacker, to assess the effects on system performance of potential defenses, and to estimate the security trade-offs of the various architectural solutions. The relational System-Aware Architectural Selection and Assessment methodology is composed of six steps; each step having a well-defined goal, required deliverables, and responsible team(s) for that stage. Phase 1 focused on a manual approach to implementing these steps. The manual processes developed in Phase 1 include (i) a scoring system that highlights cost-benefit tradeoffs for decision makers and (ii) a method based on influence diagrams designed to aid decision makers in understanding ways in which uncertainty could be introduced into an attacker's outcomes. Phase 1 methodology was successfully applied to the UAV surveillance system, both to develop the concepts through case studies and as a practical method for selection of research targets for the attack and defense teams. From these applications, it became apparent that while the methods left participants with a clear sense of having successfully explored the design space, the manual effort they required would not scale well with system complexity. In Phase 2 we began to focus on developing methods for automating the processes for evaluation of the architectural selections of critical system functions to be protected on a cyber-physical system. The Phase 2 approach is based on exploiting modeling paradigms and open source or commercial software. Specifically, the System-Aware Architectural Selection and Assessment methodology is recast in terms of use of model-based systems engineering tools, such as SysML, and attack tree tools, such as the commercial package SecurItree. Much of the Phase 2 effort was directed at developing concepts and software to integrate systems models with attack trees to facilitate an iterative architectural design process in which decision makers could "flip" back and forth between defender and attacker views of the system or mission.

The Phase 2 activity culminated in a workshop in November 2014 with participants from 10th Fleet Cyber Command and the Johns Hopkins Applied Physics Lab. The goal of the workshop was for participants to discuss the complexities of the decisions and tradeoffs inherent in choosing a defensive architecture as well as to introduce a methodology and toolset being designed to support decision makers. The format was interactive; participants engaged in the design of a defensive architecture for aspects of the UAV system, including the video surveillance mission. Highlights of the exercise included the use of the architectural scoring tools and an introduction to SysML and cyber-attack tree support tools, such as SecurITree, that can represent the system from an integrated defender and attacker perspective. Discussion focused on opportunities for systems-aware cybersecurity deployment and how future versions of the tools might best support decision makers.

The following sections review both the Phase 1 activities in architectural selection. First, we address the manual and the facilitated methods accomplished in the project. Then we describe the progress towards the automation of the System-Aware Architectural Selection methods combining system modeling techniques (such as SysML) and attack tree vulnerability analysis tools (such as SecureItree) into an integrated and automated method for identifying critical system functionality to protect. The automated methods are further described in Part 4 of the report.

3.1.1 Architectural Selection and Assessment

3.1.1.1 Definitions

The System-Aware cyber assessment methodology described here was designed to be an iterative process that relies on inputs from a range of stakeholder communities. In order to ensure that the information being used is as accurate and certain as possible, it was imperative to ask individuals questions that were appropriate to their backgrounds and areas of expertise. This is accomplished by initially dividing the stakeholders into three distinct groups:

Red Team - The red team is made up of individuals with knowledge of cyber-attacks and potential threat agent classes. Their work is focused on developing candidate attack vectors and assessing the effectiveness of the proposed design patterns.

Blue Team - The blue team consists of designers and users of the system being protected. Their responsibilities include identifying and prioritizing the critical system functions to protect, as well as determining which security design patterns can be implemented on which system functions.

Green Team - The green team, which is comprised of experts in system cost analysis and adversary capability, analyzes costs, to both the attacker and defender, for candidate architectural solutions.

3.1.1.2 Methodology Process Steps

Step 1: Define the Variables and Relationships within the System to be Protected

The initial step of the methodology is focused on framing the problem to ensure that all participants in the process are on the same page regarding the system to be protected. The process begins by identifying the critical functions of the system and defining the variables and influence relationships within that portion. Step one is to be performed by the blue team and is intended to outline the expected functionality of the system with minimal defensive strategies implemented. At this point, a system influence relational diagram is constructed using directed acyclic graph (DAG) notation. This diagram is created for the system without the consideration of a cyber-attack to ensure that everyone involved in the process is in agreement on the most basic structure and components before the additional complication of an adversary.

Step 2: Identify the Possible Paths an Attacker Could Take to Exploit the System

Step two introduces one of the issues that make this specific problem unique: an intelligent adversary. While the system influence relational diagram represents a system where success may be compromised by random failures, the cyber security architecture selection problem introduces concerns where the decisions made by an active player in the system can also compromise mission success. In step two, the red team is tasked with constructing an attack tree for the system functions identified in step one. By looking at the system from the perspective of an adversary, attack trees can be utilized to understand the possible paths an attacker could take to exploit a specific feature of the system.

Step 3: Determine the Subset of Attack Actions Most Desirable to an Attacker

Considerable analysis can be conducted after the construction of an attack tree. However, rather than focusing on quantitatively calculating the probability of success for a specific attack path, as is typically done in attack tree analysis, the analysis included in this framework considers a more qualitative, abstract metric space. In step three, the green team develops a set of variables that can be used to assess the difficulty of a particular attack path. These variables are called behavioral indicators and can include, but are certainly not limited to, resources such as technical ability, time, manpower, money, equipment, facilities, presence of an insider, and access to system design information. These variables are used to make two separate types of judgments: leaf node assessments and adversary profile construction.

Step 4: Identify Appropriate Defensive Actions and Their Impacts on the Attacker

After the red and green teams have identified the actions that an adversary would need to take to successfully execute an attack and the subset of those that are most attractive to a particular adversary, the blue team can then determine which of their existing defensive actions may be appropriate. The relational methodology relies on the assumption that a portfolio of design patterns has already been developed—either by previous blue teams or by an external group no longer involved in the process. If the current blue team was not responsible for developing the set of design patterns, it is assumed that they have access to the portfolio and the have the necessary knowledge regarding the meaning of each design pattern.

The goal of step four is to select design patterns from the existing portfolio that could be implemented to make the actions captured in the leaf nodes of the attack tree less desirable to the attacker. This can mean increasing the difficulty, cost, or probability of detection to the adversary or lessoning the consequences felt by the defense in the case of a successful attack.

Step 5: Evaluate the Impacts of the Selected Potential Actions on the Defense

While step four captures the design patterns' impacts on the adversary, step five transitions to evaluating how those same choices impact on the performance of the system to be. The green team is able to apply their second class of intelligence information here: cost analysis estimates for the defensive solution choices. At this point, each of the design patterns selected in step four is evaluated in regards to implementation cost, lifecycle cost, and collateral system impacts. The green team is responsible for estimating the monetary cost of a solution, but the blue team also adds input on a solution's collateral system impact here. The blue team performs the evaluation of the solution's collateral impacts since they have knowledge regarding the system, how it will be used, and what impacts are unacceptable. Any solutions that are deemed to be beyond the allocated budget for System-Aware security or introduce unacceptable impacts on system performance can be eliminated from further analysis at this point.

There is one deliverable for this step: a reduced list of possible defensive choices, filtered from the original existing design pattern portfolio, to only those that increase the difficulty for the considered attacker while still remaining at an acceptable impact to the defense.

Step 6: Weigh the Security Trade-offs to Determine Which Architectural Solutions Best Reverse the Asymmetry of a Potential Attack

The goal of the sixth and final step is for all three teams to participate in a collaborative discussion regarding the security trade-offs that exist with the potential choices determined in step five. While each defensive strategy remaining after step five provides some potential security benefit, has an acceptable impact on the system being protected, and fits within the allocated budget the exact mixture of security to defense to budget varies by solution.

3.1.1.3 Vulnerability and Threat Analysis Process

When trying to protect a UAV or UAS from a cyber-based attack, important questions arise when identifying priorities for potential threats, purposes, consequences and level of effort to achieve them: Which UAV systems and functions, if compromised, can lead to significant disruption? What UAV components or system configurations are inherently vulnerable to classes of cyber-attack? Where can these threats originate?

One approach to answering these questions is to begin with a cyber-attack classification schema that allows one to reason about vulnerabilities and impacts in a structured way. While most schemas in other domains are one or two dimensional in nature, cyber-attacks on cyber physical systems such as UAV systems are usually multi-dimensional owing to the fact that the exploits, deployment, and effects of the attacks usually involve a multi-vector approach that can occur anywhere along the lifecycle of the UAV. Our research aimed to develop a structured methodology to identify potential vulnerabilities, reason about the attack surfaces that exploits may use, and rank the impacts of potential cyber-attacks to allow more systematic development of cyber defenses.

An architectural selection framework for System-Aware cyber enhancement was developed and applied to the autopilot system in the project. We provide an overview of activities for cyber threat analysis efforts in this section that supports the overall cyber enhancement architectural selection process:

1. Define the system functions and relationships between those functions within the system.
2. Identify the critical system functions and subsystems.
3. Identify of potential cyber-attacks.
4. Determine the subset of attack actions most desirable to an attacker.

3.1.1.4 System Functions and Their Interrelations

The purpose here is to develop an influence graph between major systems such that functional dependencies between systems can be reasoned about.

By studying the general architecture of the autopilot in Figure 7, we can see a natural grouping of relationships for the autopilot into four categories:

- **The Controller**: The onboard processor executes all of the control laws, flight director functions, management of INS (inertial navigation system), GPS, actuators, and the communication links. The controller is composed of those functions represented by the red circle in Figure 7. The flight controller requires inputs from the sensory subsystem state estimator (e.g. INS, GPS, altitude, and speed) to regulate the aircraft to a desired state, speed, position and attitude. The controller also takes input from the flight director, which contains the desired trajectory reference states for the aircraft. The flight controller uses the information stored in the flight director as tracking inputs; thus the flight controller is progressively issuing actuation commands to the control surfaces to minimize the error between track references and current aircraft state and position. As such, the autopilot continuously flies the aircraft to each geographical waypoint in succession. Attacks directed to the hardware and software of the flight controller can affect the behavior of the flight controller so that it does not perform its function as intended.
- **Sensory and Measurement Subsystem**: The sensory subsystem (shown as the blue circle in Figure 7) provides all of the sensed vehicle state information needed by the controller to maintain stable flight. The functions in this system include the INS, which provides vehicle 3-axis accelerations, angles, and velocities; GPS which provides geo-reference position and velocities; magnetometer which is used to sense heading direction. Thus the total vehicle state is (ϕ, ϑ, ψ v_e, v_n, v_d, a_x, a_y, a_z, and heading$_)$). The total sensor readings combined with the GPS information are sensed by the controller on regular time intervals (every 100ms). Examples of attacks against the sensory subsystem include false data injection attacks to manipulate sensory data, vehicle/system component state data manipulation, and navigational waypoint data manipulation.
- **The Communication System**: The communication system is responsible for (1) transmitting commands to the UAV to alter its flight path and (2) receiving telemetry information about the UAV in flight (the communication system is identified by those components in the green circle in Figure 7). The command signals to control the aircraft are transmitted by the operator via a line of sight communication transceiver. The ground station communication link operating frequency is usually in one of several designated bands (900 MHz or 2.4 GHz are common). Various signal modulation methods are used to encode the link channels. Various channels are allocated for each command or telemetry class; i.e., pitch, roll, yaw, and throttle will be on a separate channel than GPS. After the onboard receiver decodes the signals from the ground station transmitter, the signals are converted to digital commands, processed by the onboard main processor. Attacks that target the communication system could affect both the aircraft and the command/control station. Telemetry data can be spoofed from the UAV, command information can be intercepted an altered, disabling of the communication link, etc.
- **Gimbal Pointing Camera system**: UAVs are predominantly used as Intelligence, Surveillance, and Reconnaissance (ISR) platforms carrying sensor payloads such as EO/IR cameras, synthetic aperture radar, signals intelligence systems, and others. The purple circle shown in Figure 7 encapsulates the onboard gimbal mounted camera of the UAV. The gimbal is capable of target tracking, scene steering and electronic image stabilization. The gimbal system features an onboard processor to control the stabilization effectors, a VPS, and a communication link to

send images to ground station and to the ViewPoint operator station. The ViewPoint operator station is capable of integrating with a moving-map, real-time mosaicing, target tracking, and video recording functions.

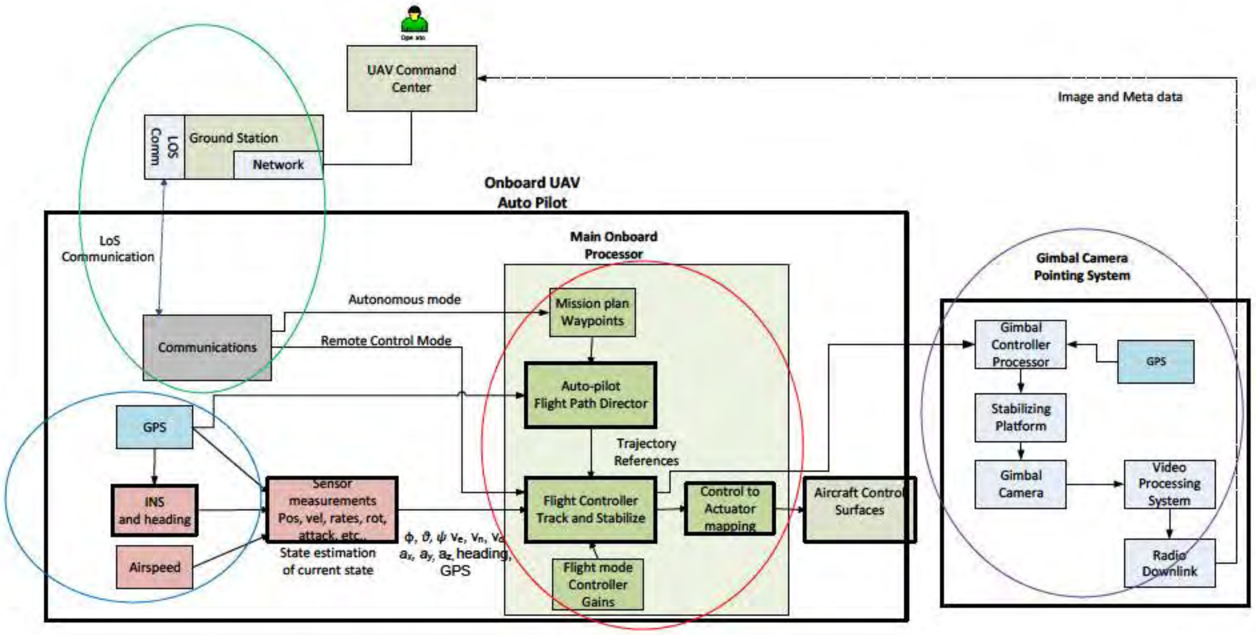

UAV Command Center
Image and Meta data
LOS Comm
Ground Station
Network
Onboard UAV Auto Pilot
LoS Communication
Main Onboard Processor
Gimbal Camera Pointing System
Autonomous mode
Mission plan Waypoints
Communications
Remote Control Mode
Auto-pilot Flight Path Director
Gimbal Controller Processor
GPS
Stabilizing Platform
Trajectory References
Gimbal Camera
Video Processing System
INS and heading
Sensor measurements Pos, vel, rates, rot, attack, etc.
Flight Controller Track and Stabilize
Control to Actuator mapping
Aircraft Control Surfaces
State estimation of current state
ϕ, ϑ, ψ Ve, Vn, Vd ax, ay, az, heading, GPS
Airspeed
Flight mode Controller Gains
Radio Downlink

Figure 7. UAV Onboard Systems showing the four major system groups.

To understand the relationships between the major subsystems, we utilize an *influence diagram*, which is a type of *DAG*. DAGs provide value in situations where a system is characterized by a large number of inter-dependent functions/variables that have highly coupled process coordination. Understanding the possible attack scenarios is dependent on understanding the interrelationships among these coupled functions. For this reason, they work well for considering a system of this scale and have been used for a variety of applications in the safety and reliability fields.

As shown in Figure 8, a DAG includes a set of *nodes* and a set of *edges* connecting the nodes. In the system influence diagram shown in Figure 8, nodes represent a *functional resource* within the system. These can be hardware or software components, interfaces, or external factors, all of which have system functionality and can influence the outcome of the system service or behavior. The edges connecting the nodes represent the influence relations between the nodes. If two nodes are connected that means one node is influenced by the other node in order to provide expected service to UAV system. The arrow on the edge connection signifies a *provides relation*. The accepting node signifies a *requires relation* from the edge. Similarly, if two nodes are not connected, the functionality of one node does not have an influence on the other. While a DAG alone overlooks a critical aspect of the problem at hand (the presence of an adversary), its construction enables the team to reach a common understanding of the system.

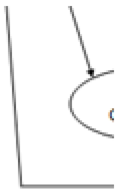


Figure 8. Influence diagram for used to understand the relationship between the UAV subsystems used for navigation.

Figure 8 shows that the output of the *Aircraft Navigation* (i.e., the success or failure of the aircraft navigation function) is dependent on three major factors: (1) the actions of the human operator, (2) the functionality of the autopilot hardware/software, and (3) the weather conditions where the platform is currently operating. In turn, the autopilot function is dependent (i.e., influenced) by a number of its upstream nodes. These include state estimates from the sensor subsystems, pre-flight configurations, stored waypoints, communication links, INS, GPS, etc. All of these upstream nodes, if compromised by a cyber-attack, may alter the navigation of the UAV. For instance, if the GPS receiver is compromised in such a way that the latitude and longitude coordinates are offset then the navigation tracker will think the UAV is in a location where it is not and attempt to move the UAV to the desired waypoint. That is this type of cyber-attack would cause the UAV to divert from its planned path.

Similarly, the diagram shows that the status of the operator display is influenced by static information; e.g., maps that are stored in the software and variable information of the state estimates which are collected on-board the platform. In turn, the information showed on the display influences the actions of the human operator.

Figure 9 shows the influence diagram for the gimbal camera pointing system. The subsystems of interest in this diagram are the camera control processor, GPS, and the sensors and effectors. The camera control processor executes software (SW) to implement functions such as, pan-zoom-tilt (PZT), auto-tracking, point-of-interest tracking, etc. The GPS receiver provides the necessary geo-reference data to the control processor and camera to locate and track objects of interest. The sensor and effector group provides motion-stabilization to the mounted camera during flight. These three factors provide the greatest influence to the success or failure to the surveillance mission.

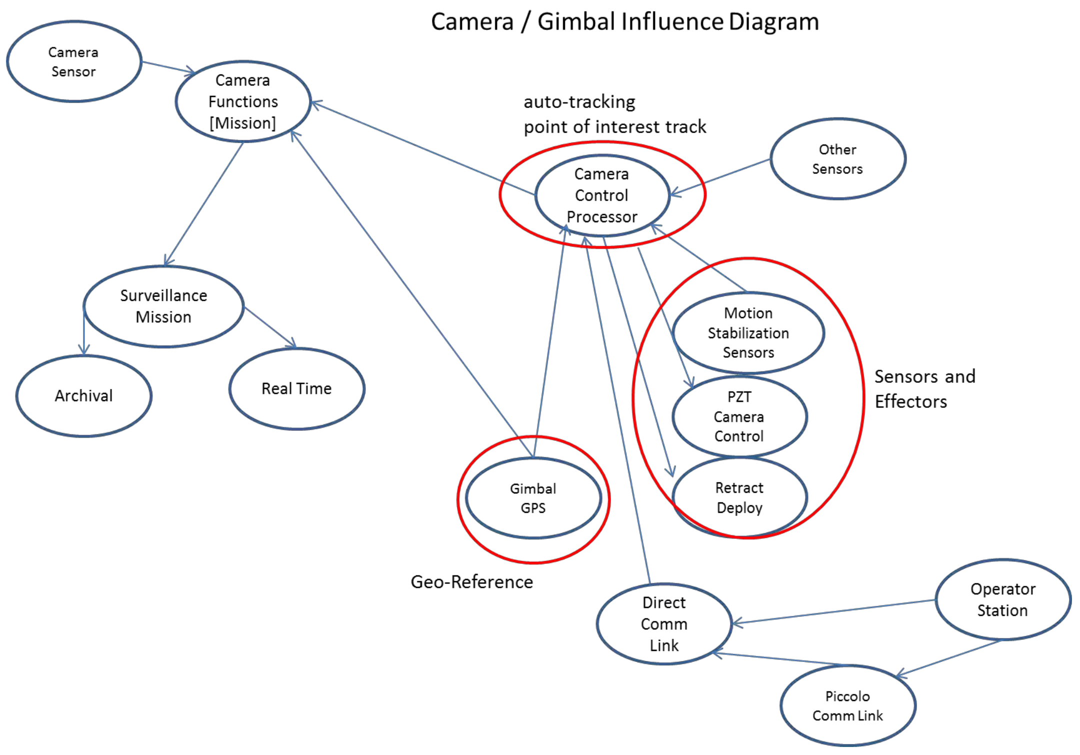


Figure 9. Influence diagram used to understand the relationship between the UAV subsystems used for gathering surveillance data.

With a firm understanding of the UAV system functions and how their interrelationships can influence or affect the UAV navigation, we can now transition toward identifying critical systems onboard the aircraft.

From Figure 8 and Figure 9, three major subsystems have been identified for further analysis:

- Autopilot subsystem.
- GPS subsystem.
- Gimbal camera pointing systems.

3.1.1.5 Identifying and Classifying Potential Cyber Attacks

To support this effort we developed a cyber-taxonomy to assist the red team and blue team members to think broadly about the origins, effects, and extent of potential cyber-attacks on the UAV. A taxonomy or classification schema allows practitioners to have a common basis of understanding. It also allows one to systematically reason about cyber-attack characterization as *classes*. In doing so the analysis of cyber-attacks is more organized and easier to transfer to other cyber defense engineering practices. Our taxonomy was developed to support the following analysis inquiries required of the System-Aware cyber security framework:

- What are the different ways of perpetrating an attack against UAV systems?
- What kind of damage or consequence can these attacks cause?
- What are the challenges in preventing such attacks?
- What are vulnerabilities that allow the attack to manifest?
- What are potential propagation channels of the cyber-attack?

Figure 10 shows the taxonomy model. Each node at level 2 of the tree (the tree in Figure 10 contains 9 levels) can be thought of as a dimension in an ordinal structure. That is, each dimension has a specific place in the order of the taxonomy. The dimensions of the model include the *objectives of the attack, propagation means, origin of attack, actions of the attack, vulnerabilities exploited,* and *target resource, effects and consequences.* Here the order is organized around the following chain of inference:

> An attack <u>OBJECTIVE</u> by means of <u>PROPAGATION</u> from a lifecycle <u>ORIGIN</u> using malicious <u>ACTIONS</u> exploiting a <u>VULNERABILTY</u> on a <u>RESOURCE/TARGET</u> can change system <u>EFFECTS</u> that have system <u>CONSEQUENCES</u>

Below each dimension are sub-dimensions or categories that characterize the parent class dimension with respect to the domain of applicability. This classification schema recognizes that these sub-dimensions or categories can be modified to fit other domains; e.g., a cyber-attack on a power grid may have different sub-dimensions than a UAV (i.e., the target dimension for the power grid would be substations and control centers). In addition, new types of attacks that may require new sub-dimensions can be added to the schema without altering the basic dimensions.

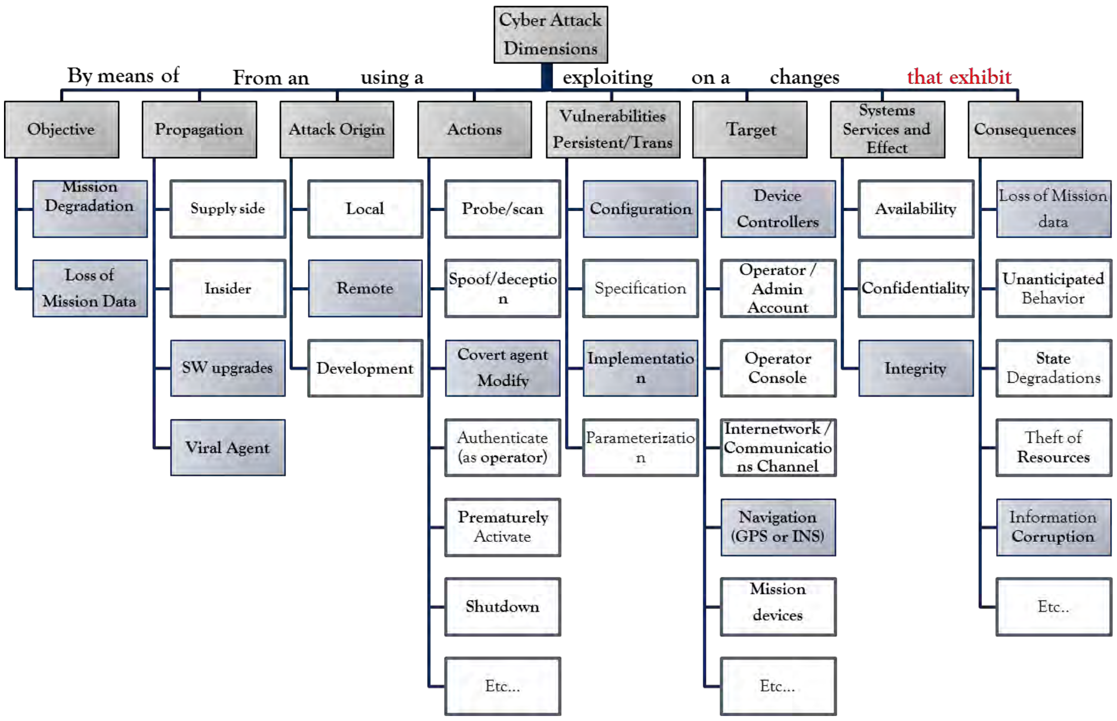


Figure 10. Cyber-taxonomy.

3.1.1.6 Selection of Cyber Attacks to move forward

Based in part on the cyber-attack profiling detailed in Barbara Luckett's 2013 thesis, and the categorization of cyber-attacks by the taxonomy method described above, we selected several classes of cyber-attacks for more detailed analysis and to carry forward to the System-Aware cyber test-bed phase and ultimately into flight evaluations for our UAV surveillance platform:

- Parameter-Based System Attack
- GPS System Attacks
- Gimbal System Attacks
- Hardware Security Against Manufacturing and Design Attack

The selection of these attacks is based in part on (1) how each cyber-attack is uniquely different and thus stresses the System-Aware cyber security methodology, and (2) how the application of each cyber-attack may result in different effects on the overall UAV system operations. Before we discuss the specific cyber-attack profiles, we first introduce the concept of an *attack surface*. While the taxonomy described above is beneficial in postulating about the classes of cyber-attacks, it is not intended to describe in detail the specific mechanisms or vectors that an attack uses to penetrate the system. In order for our emulated cyber-attacks to reflect actual cyber-attacks, we need to ensure that realistic attack surfaces exist for the emulated cyber-attacks in the UAV.

3.1.1.7 Understanding Attack Surfaces

The attack surface of a software environment is the sum of the different points (the attack vectors) where an unauthorized user (the attacker) can try to enter or extract data from an environment. The model in Figure 11 illustrates the concepts of attack surfaces on several important points. First, successful cyber-attacks usually require several attack surfaces to be breached for success. The second notion is reachability. Reachability describes the depth or breadth of the influence effects of the attack. In this case, the red arrow indicates an attack that deeply penetrates all layers to achieve its objective. The third notion is entry and exit points. Entry points define the places where data is inputted into the system; thus providing a means for ingress into the system by cyber-exploit. Entry points are associated with channels. Channels are means for moving or observing information into a system either directly or indirectly. A channel could be a network port, an unused debug port, or a wireless snooper. The exit points define where data or control information can be acquired from a system. Finally, resources that a device uses to input, move, process, and output data are part of the attack surface. Resources have entry and exit points, processing channels, and storage.

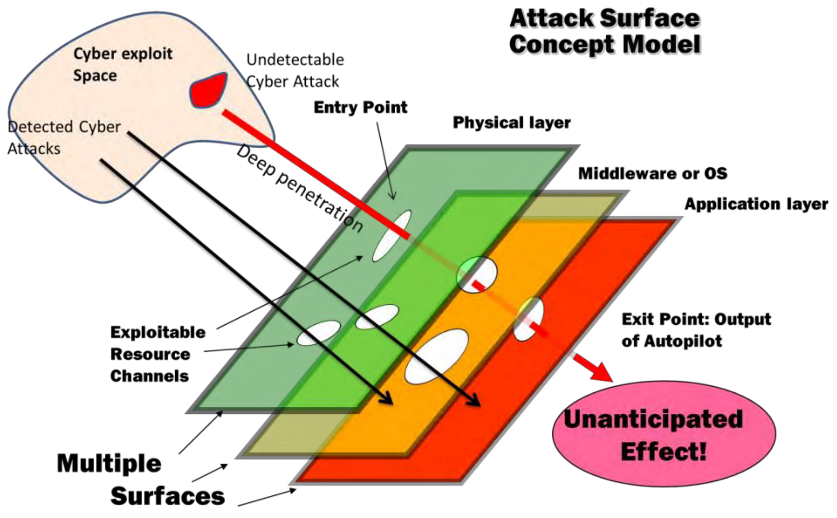


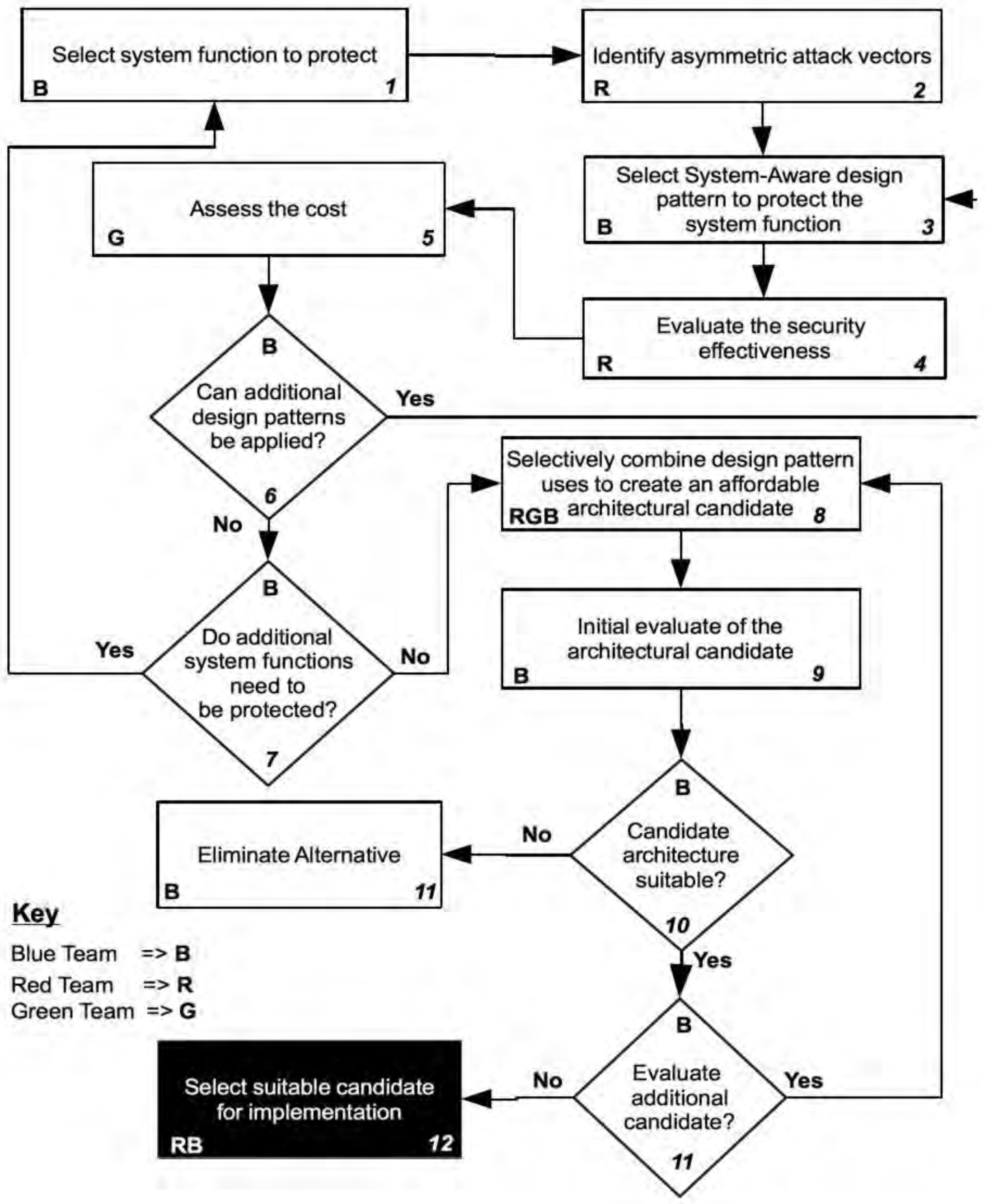
Figure 11. Attack Surface Concept Model.

3.1.2 Cybersecurity Architectural Selection – a UAV Case Study

Earlier sections have outlined a model-based approach for identifying possible attack vectors that could be used to exploit the UAV system being protected, those system functions that would benefit from System-Aware protections, and applicable System-Aware Cybersecurity design patterns to provide those protections. However, due to limited resources (e.g., money, power, size, and weight) it may not be possible to implement a comprehensive solution. As a result, decision makers will be required to decide when and how to best utilize their limited resources to realize potential solutions to mitigate the identified threats. In initial tool set was developed in this project to address a facilitated approach for the identification and selection of key architectural functions that system owners would want to protect on their systems. Figure 12 shows a flowchart for the architectural selection tool that can be used to aid decision makers in how best to expend their limited resources on the solutions identified in previous sections. This process includes the selection of those critical system functions to be protected using System-Aware security from those identified using the model-based approach previously outlined, relative ranking of the effectiveness of System-Aware solutions at addressing asymmetric attack vectors, selection of design patterns to protect those system functions from potential cyber attacks, the integration of those system functions and System-Aware design patterns into candidate architectures, and finally an evaluation of how the architecture affects the asymmetry between potential attackers and the system being protected. As described in Section 3.1.1.1, each of these steps is to be carried out with the assistance of one of three teams:

- *System design (Blue) team* – Members of this team includes those that understand how the system was designed, implemented, and operates
- *Cyber attack (Red) team* – Members of this team are knowledgeable about the resources necessary to create and design exploits
- *Cost Analysis (Green) team* – Members of this team are responsible for determining the costs of designing, implementing, and integrating the selected System-Aware security services

The remainder of this section details how steps 1 through 11 (i.e., the selection of suitable architectural candidates) are accomplished supported by the prototype tool developed in this project. Section 3.1.3 outlines the prototype implementation of the tool for the selection of suitable architectural candidates, as well as discusses its application to a use case – the systems on board the experimental UAV surveillance platform.



Select system function to protect
B 1
Identify asymmetric attack vectors
R 2
Select System-Aware design pattern to protect the system function
B 3
Assess the cost
G 5
Evaluate the security effectiveness
R 4
B
Can additional design patterns be applied?
6
Yes
No
Selectively combine design pattern uses to create an affordable architectural candidate
RGB 8
B
Do additional system functions need to be protected?
7
Yes
No
Initial evaluate of the architectural candidate
B 9
B
Candidate architecture suitable?
10
No
Yes
Eliminate Alternative
B 11
Key
Blue Team => B
Red Team => R
Green Team => G
B
Evaluate additional candidate?
11
No
Yes
Select suitable candidate for implementation
RB 12

Figure 12. Flowchart illustrating the Cybersecurity Architectural Selection framework. This framework enhances the previously outlined identification and evaluation framework by including a process to help design and select one or more architectural candidates from the potentially vast set of possible System-Aware architectures given a limited set of resources. Each step is also marked with the team(s) designated to perform that step.

3.1.2.1 Selection of System Functions for Protection

The first step in designing a System-Aware architecture is to identify which of the system functions will be protected using System-Aware Cybersecurity. This includes, (1) identifying those system functions that could possibly benefit from System-Aware Cybersecurity, and (2) determining the relative importance—from a cyber security viewpoint—of protecting those functions from a cyber-attack. This importance is relative because it is based on a comparison of every system function to every other system function. Furthermore, as the importance is relative, it enables the ability for all system functions to be ranked. As discussed in Section 3.1.4.4, this ranking will be used to compose a subset of all identified system functions into a set of the most important functions to protect. It is emphasized that the relative importance assigned to protecting a system function is based on its contribution to achieving the mission objectives of the system and not on its susceptibility to attack.

The identification of system functions is accomplished using the methods outlined in earlier sections. The ranking of their security importance should be performed with the aid of a system design team (i.e., Blue Team), as it possess the knowledge necessary to determine functions are exposed to types cyber-attacks System-Aware security is designed to address and how those functions are used to achieve the system's mission objectives.

3.1.2.2 Identification of Asymmetric Attack Vectors

After the system functions to be protected have been identified, each of those functions is assessed to determine possible asymmetric threats. An asymmetric threat is one that, with only a minimal analysis of the system function to be exploited, could be designed, developed, and maintained utilizing a small amount of resources, and has the potential to severely compromise or damage the system to be protected. In the absences of System-Aware security, such threats are potentially difficult to detect and deflect. This step is to be performed by a red team using the information provided by the model-based approach outlined in previous sections.

3.1.2.3 Selection of System-Aware Design Patterns

Once all of the system functions that could benefit from System-Aware security have been identified, it is then necessary to select System-Aware design patterns to be considered for protecting those functions. A design pattern is selected if it provides one or more of the following

1. Makes it significantly more difficult for the compromised system function to cause damage to the system or compromise the system's ability to complete its designated mission objective
2. Improves the robustness of the system function by either
    a. Preventing a compromised system function from taking certain damaging actions
    b. Allowing for the restoration of a compromised system function into a non-compromised—and possibly less capable—state
3. Increases the likelihood of identifying when the system function has been compromised and/or information about the source of compromise (i.e., identifying attributes about the adversary)

Since determining the efficacy of applying a System-Aware design pattern to a given system function requires knowledge of how that function operates, as well as how it integrates into the overall system design, the selection of System-Aware design patterns is performed with the aid of the system design team (i.e., Blue Team). Note that this selection does not consider issues associated with attackers' developing and executing exploits. This is deferred for a subsequent stage in the analysis process.

3.1.2.4 Determining the Security Effectiveness

Given a set of system functions, each with a set of System-Aware design patterns, it is then necessary to determine the potential efficacy (i.e., the effective security) each System-Aware design pattern affords to the function it has been selected to protect. Several approaches exist. For example, one approach is to utilize structured arguments to organize a large body of evidence into well-structured rigorous arguments about the effectiveness—from a cyber security viewpoint—of a proposed solution. The architectural selection method outlined in this section outlines a streamlined process for creating a manageable set of System-Aware architectural candidates to be evaluated. This screening process involves the identification and evaluation of all system functions that could benefit from System-Aware cyber security, as well as all possible System-Aware design patterns that could be utilized to protect those functions. This is accomplished by assigning an integer valued score to a single System-Aware design pattern protecting a single system function based upon two criteria

1. The System-Aware design pattern's ability to increase the complexity, cost, and time for a hypothetical adversary to develop an exploit for the system function being protected
2. The ability to decrease the probability that an attack against the system function being protected will be successful

This simplified analysis assumes that every combination of System-Aware design pattern and system function provides the system with its own independent effective security; i.e., there are no (dis)economies of scale exist. In addition, this method assumes that the security effectiveness of a given System-Aware architecture is simply the sum of the security effectiveness scores for all of the included combinations of System-Aware design patterns and system functions.

Such an evaluation requires knowledge about the resources necessary to design and develop exploits; as such, this step is to be performed by a cyber attack team (i.e., Red team). In addition, the cyber attack team should perform its evaluation without knowing the relative rank ordering of system functions performed by the system design team or the reasoning behind the selection of System-Aware design patterns.

3.1.2.5 Determining the Resources Necessary to Develop the System-Aware Architecture

System-Aware architectures are evaluated in terms of their security effectiveness, collateral impacts on system performance, and costs to design, implement, and maintain. Thus, candidate System-Aware architectures cannot be designed or evaluated without a method for estimating their potential costs. For the Cybersecurity Architectural Selection approach resented here, these costs are to be estimated by a special cost analysis team (i.e., Green Team) for all of the System-Aware design patterns proposed for all of the identified system functions. As was the case for security effectiveness, these costs represent the costs of designing, implementing, and maintaining a single design pattern for a single system function. This analysis assumes that no (dis)economies of scale exist, and the total cost of a System-Aware architecture is simply the sum of the cost for each of the selected combinations of

System-Aware design pattern and system function. This analysis is to be performed by a specialized cost analysis team.

It is observed that the costs analysis team can also evaluate the costs for an adversary to analyze, design, implement, and maintain counter measures to the proposed System-Aware security measures. This information can be utilized to assess the amount of asymmetry, if any, between the costs to implement System-Aware security measures and the costs to overcome those measures. In turn, the relative levels of asymmetry can then be used to help evaluate the security effectiveness of the candidate architectures. It is noted that this asymmetry is dependent on the perceived resources of the adversary versus the system being defended; e.g., assume there is an adversary with a small amount of resources and a System-Aware architecture is being designed to protect a critical large system that was expensive to design and maintain. In this circumstance the adversary only has a small amount of resources to overcome any System-Aware defensive measures that are developed. However, the System-Aware architecture can consume a larger amount of resources, but this can still be relatively small compared to resources being protected. Thus, it is the costs relative to the resources available that determine the asymmetry.

Collateral impacts on system performance is a rough estimate provided by the system design team and are used solely to eliminate System-Aware design patterns that would unacceptably degrade system performance. Collateral impacts are not used more extensively for the purposes of designing candidate architectures as only the system owner can decide whether the potential security benefits outweigh the loss in system performance, and due to the potential for mitigating solutions (e.g., increasing the system's processor speeds or the amount of available memory). During architectural evaluation, collateral impacts can be estimated using more rigorous methods as desired.

3.1.2.6 Selecting Architectural Candidates

Once all of the system functions that could benefit from System-Aware Cyber Security have been identified, System-Aware design patterns have been selected and their potential security effectiveness have been evaluated, and their costs and impacts have been assessed, it is then necessary to select architectural candidates for evaluation. However, while selection and evaluation can be treated as separate activities, it can be beneficial to combine these steps. In the architectural scoring framework this is accomplished through a combination of automated decision tools and user interaction. Automation is utilized to generate a small set of candidate architectures. User input is utilized to provide constraints and to create candidate architectures by modifying the candidate architectures with the support of automation tools. The candidate architectures are created by utilizing the relative importance of system functions, as determined by the system design team, the potential security effectiveness afforded by applying System-Aware design patterns to system functions, and the costs of implementing the selected System-Aware design patterns. In addition, these criteria can be used to help users modify these architectures; e.g., users could replace a subset of System-Aware design patterns chosen by the automated system with more expensive System-Aware design patterns that offer a greater degree of effective security. Section 3.1.3 provides a discussion of a prototype architecture that illustrates one approach to using automation and user interaction to generate candidate architectures. The result of this process is a set of suitable architectural candidates.

3.1.2.7 Limitations of the Proposed Enhanced Architectural Scoring Framework

While the decision support system outlined in this section provides several enhancements to the System-Aware architectural evaluation process, it still offers room for improvement

1. The enhanced scoring framework assumes that no relationship exist among the System-Aware design patterns applied to protect the system function; i.e., no conflicts or redundancies are present

2. The security effectiveness and resource cost only apply to a single System-Aware design pattern applied to a single system function this allows them to be assessed independently and can be combined in a simple additive manner (i.e., linear) with the security effectiveness and resource cost of other design patterns to obtains the security effectiveness and resource cost of the final architecture; i.e., no economies—or diseconomies—of scale exist

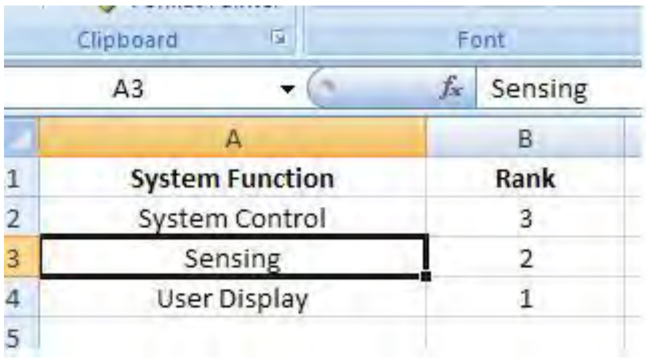
System-Aware architectures that have the same costs and security effectiveness are equally desirable to a decision maker regardless of the number and particular System-Aware design patterns implemented and system functions protected; i.e., an architecture with a single highly secured system function based on the integration a number of design patterns, can be as valuable as an architecture with several system functions protected, but with fewer design patterns utilized to protect each function.

3.1.3 Prototype Architectural Assessment Support Tool

To help in the creation of the Cybersecurity Architectural Selection framework described in section 3.1.2, a prototype decision support system was constructed using Microsoft Excel. There were two principle goals in developing this prototype system. First, was to develop algorithms for automatically generating architectural candidates. Second, was the creation of tools to help users explore the set of all candidate architectures by making adjustments to the automatically generated architectures. This also included the creation of tools and visualization aids to help users compare and contrast the candidate architectures. Finally, while these make up the main contribution of the prototype decision support system, it is noted that the prototype is fully functional; i.e., it supports all of the steps laid out in section 3.1.2.

3.1.4 Selection of System Functions for Protection

Figure 13 shows how the prototype system supports the selection of system functions for protection for a simple example system. As can be seen, three system functions have been identified and assigned a relative ranking according to their importance to protect by the Blue Team. For the system shown in Figure 13, the System Control function has been designated the most important system function to protect, and the User Display the least; i.e., a higher ranked function is more important then a lower ranked function. It is noted that while only three system functions are shown here, the prototype system can support any number of functions.



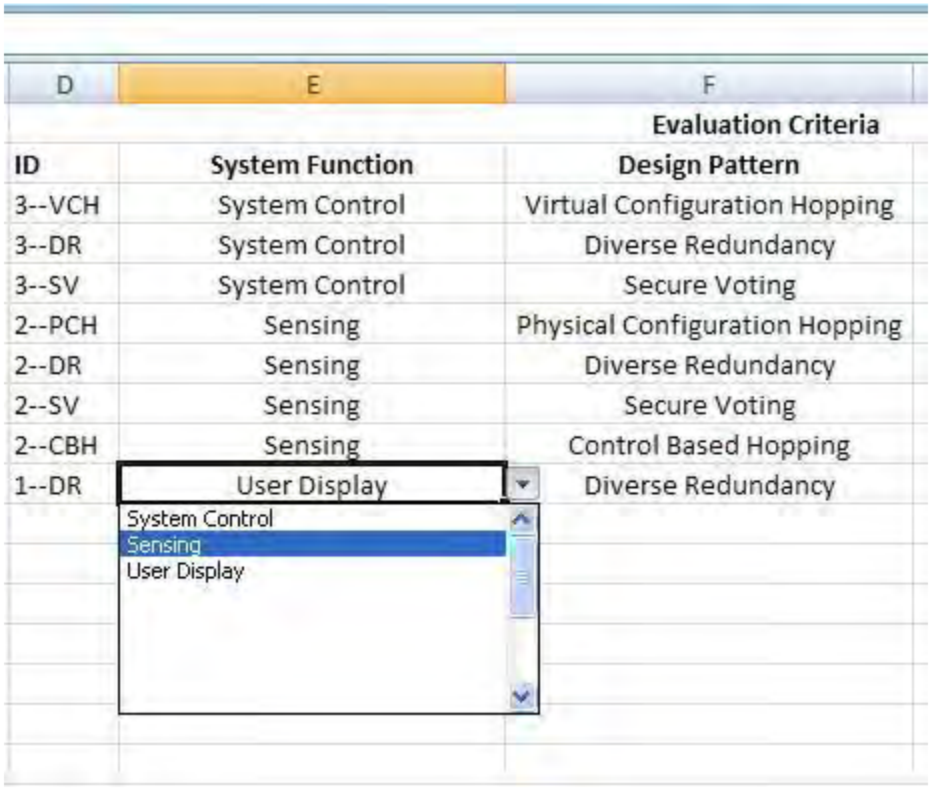
	A	B
1	System Function	Rank
2	System Control	3
3	Sensing	2
4	User Display	1
5		

Figure 13. Represents the prototype decision support system's support for selecting system functions to protect and assigning them a relative rank ordering. In this case three system functions have been identified by a Blue Team for protection and ranked. A higher rank signifies a system function that is more important to protect; e.g., in this instance the most important function to protect is the one responsible for System Control.

3.1.4.1 Selection of System-Aware Design Patterns

After the system functions that would benefit from System-Aware security solutions have been selected, System-Aware design patterns are chosen by the Blue Team to protect these functions. Figure 14 shows this for the System Functions identified in Figure 13. As can be seen, the user first selects one of the identified system functions from a drop down list. After selecting a function for security consideration, a System-Aware design pattern is chosen for that system function. Finally, each combination of system function and System-Aware design pattern is assigned a unique identifier.

The prototype system offers two methods for creating this identifier. The method illustrated in Figure 14 uses the previously designated relative rank combined with a short acronym of the System-Aware design pattern. This method is intended to produce an identifier that would allow the user to intuitively recognize the relative importance of the system function as well as the method used to protect it. However, this scheme requires all available System-Aware design patterns to be assigned a unique short hand identifier before the process is started. Depending on the number of design patterns available, this may not be practical. To address this potential pitfall, a second method is available. This method recognizes that a System-Aware design pattern can only be applied to a system function once; i.e., each combination of system function and design pattern should be unique. Hashing these unique combinations generates a unique identifier. This has the benefit of generating unique identifiers without requiring additional information from the user; however, unlike the former method, the identifiers will not have intuitively derived meaning.



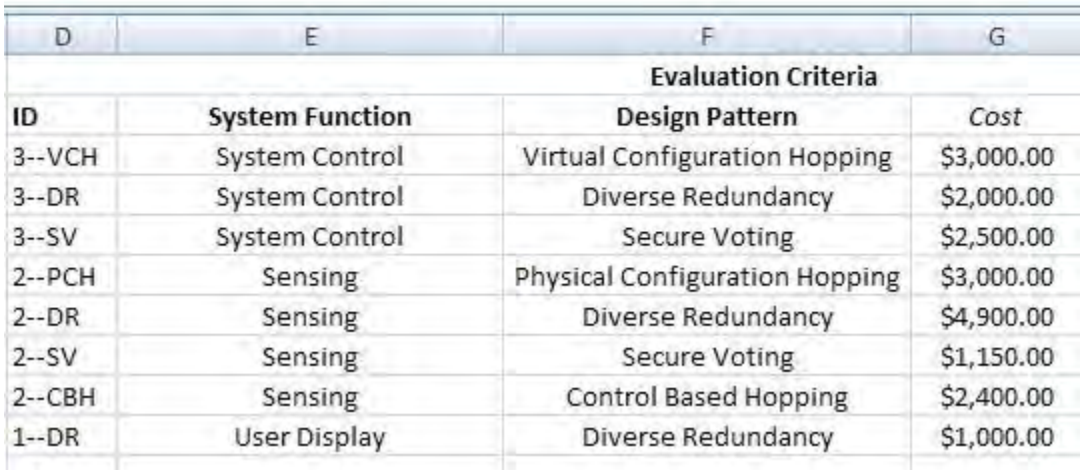
D	E	F
		Evaluation Criteria
ID	System Function	Design Pattern
3--VCH	System Control	Virtual Configuration Hopping
3--DR	System Control	Diverse Redundancy
3--SV	System Control	Secure Voting
2--PCH	Sensing	Physical Configuration Hopping
2--DR	Sensing	Diverse Redundancy
2--SV	Sensing	Secure Voting
2--CBH	Sensing	Control Based Hopping
1--DR	User Display	Diverse Redundancy

Figure 14. Illustrates how the prototype system is used to support the selection of System-Aware design patterns. For this instance the available system functions are assumed to be those presented in Figure 13. As can be seen, only those functions previously selected for protection can be selected in this step. Also note that every combination of system function and System-Aware design pattern is assigned a unique identifier. For this instance, the identifier is composed a combination of the relative rank of the system function and the first letter of every word of the System-Aware design pattern.

3.1.4.2 Determining the Resources Necessary to Develop the System-Aware Architecture

As discussed in section 3.1.2, every combination of system function and System-Aware design pattern should be evaluated independently to determine its costs and collateral impacts on the system. For the prototype architecture, it is assumed that all resources can be represented by a single monetary cost. This is more restrictive then outlined section 3.1.2, where there can be multiple costs represented in different units (e.g., money, power, and space). The decision to reduce this to a single monetary cost was made to both support a more robust set of algorithms for automatically generating architectural candidates, and to make it easier for a user to compare and contrast competing architectural candidates.

Figure 15 builds upon Figure 14 to illustrate how this is done in the prototype system. As can be seen, every combination of system function and System-Aware design pattern is assigned a single monetary cost. For the prototype system, these costs are considered to be independent.



D	E	F	G
		Evaluation Criteria	
ID	System Function	Design Pattern	*Cost*
3--VCH	System Control	Virtual Configuration Hopping	$3,000.00
3--DR	System Control	Diverse Redundancy	$2,000.00
3--SV	System Control	Secure Voting	$2,500.00
2--PCH	Sensing	Physical Configuration Hopping	$3,000.00
2--DR	Sensing	Diverse Redundancy	$4,900.00
2--SV	Sensing	Secure Voting	$1,150.00
2--CBH	Sensing	Control Based Hopping	$2,400.00
1--DR	User Display	Diverse Redundancy	$1,000.00

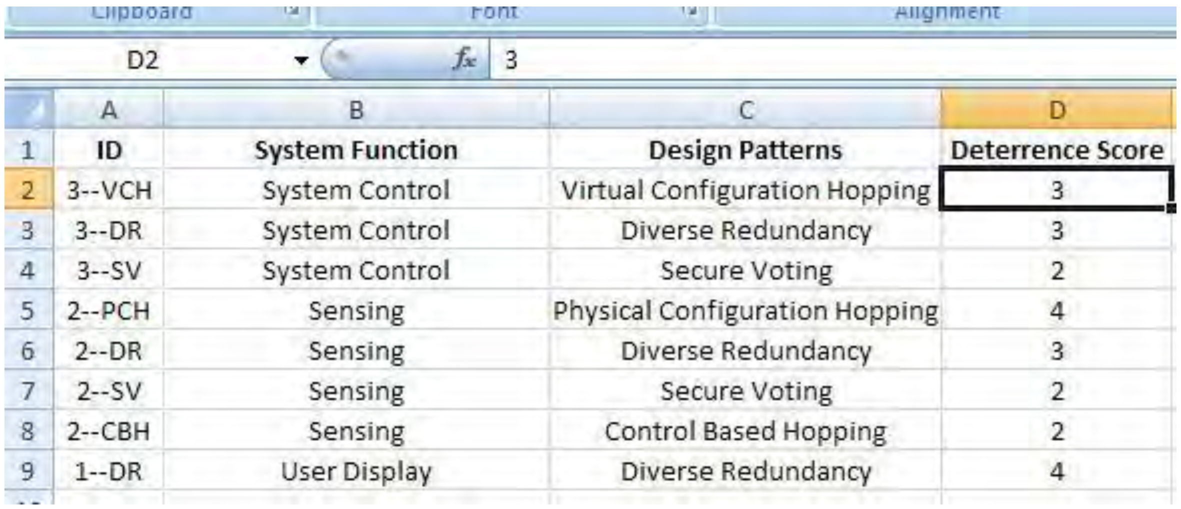
Figure 15. Illustrates how the prototype system supports the assessment of the necessary resources needed to implement a given System-Aware architecture. In this instance every combination of System-Aware design pattern and system function generated in Figure 14 is assigned a monetary cost.

3.1.4.3 Determining the Security Effectiveness

Figure 16 illustrates how the prototype system supports the evaluation of the security potentially afforded by each combination of system function and System-Aware design pattern to be evaluated. In the prototype evaluation system the ID, System Function, and Design Patterns, fields are automatically populated based on the inputs received in the previous steps The Deterrence Score is used to represent the security effectiveness potentially afforded by each combination of system function and design pattern. As noted in section 3.1.2, the security effectiveness score can be any integer value; however, for the prototype system, this score is limited to 1, 2, 3, or 4, with higher scores relating to more value. This limitation is not imposed by the evaluation system, but rather stems from the criteria used to determine the effective security score

- Score = 4, Complexity, cost, and time to develop exploits are high and the probability of a successful exploit is low
- Score = 3, Complexity, cost, time to develop exploits are high and the probability of a success exploit is high
- Score = 2, Complexity, cost, time to develop exploits are low and the probability of a success exploit is low
- Score = 1, Complexity, cost, time to develop exploits are low and the probability of a success exploit is high

The particular system in Figure 16 is built using those combinations illustrated in Figure 14.



D2 f_x 3

	A	B	C	D
1	ID	System Function	Design Patterns	Deterrence Score
2	3--VCH	System Control	Virtual Configuration Hopping	3
3	3--DR	System Control	Diverse Redundancy	3
4	3--SV	System Control	Secure Voting	2
5	2--PCH	Sensing	Physical Configuration Hopping	4
6	2--DR	Sensing	Diverse Redundancy	3
7	2--SV	Sensing	Secure Voting	2
8	2--CBH	Sensing	Control Based Hopping	2
9	1--DR	User Display	Diverse Redundancy	4

Figure 16. Illustrates how the prototype supports the assigning of security effectiveness scores to all combinations of system function and design pattern. This instance assigns scores for those combinations identified in Figure 14.

3.1.4.4 Selecting Architectural Candidates

As discussed in section 3.1.2, the selection of architectural candidates is performed through a combination of automation tools and user input. The user provides constraints that are used to guide the automated execution of algorithms in the creation of a subset of candidate System-Aware architectures. In addition, the user is then able to modify one or more of these candidate architectures to generate additional candidate architectures. This process can result in one of two outcomes

1. A set of candidate architectures to be evaluated using a more rigorous criteria
2. Select the architecture that will be implemented

The prototype system can support both outcomes; however, the prototype only supports the creation of candidate architectures and does not provide any additional support to perform a more rigorous evaluation.

3.1.4.5 User Constraints

The prototype decision support system accepts user constraints on the total amount of resources that are available for implementing System-Aware design patterns to protect system functions. Since the prototype system assumes that only one monetary resource exist, this step is presumed to be the user allocating a budget. Figure 17 illustrates the options available to the user (it assumes the resources shown in Figure 15).

- User is allowed to set a value between 0 and the sum total of the resource costs of selecting every combination of system function and design pattern (this is 19,950 in Figure 17)

- User can move a slider between the values of 0 and the sum total of the resource costs of selecting every combination of system function and design pattern (this is 19,950 in Figure 17)

The maximum budget value is automatically computed based on the inputs supplied. Furthermore, the budget values are always integer—the maximum value is always rounded up to ensure that is can cover the difference. This is done to support the automatic generation of candidate architectures.

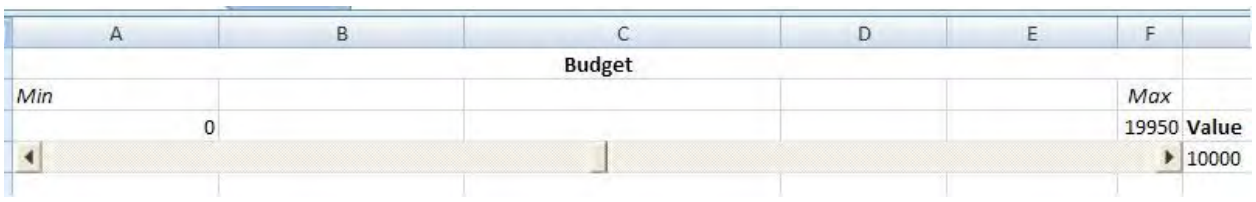


Figure 17. Illustration of the budget slider used in the prototype system. This slider is currently set to a budget of $10,000, has a maximum possible value of $19,950, and a minimum value of $0.

3.1.4.6 Automated Support to Generate Candidate Architectures

After a maximum budget has been set and System-Aware design patterns have been selected, their security effectiveness evaluated, and their cost determined for each of the system functions identified as possibly benefiting from System-Aware security, an automated decision support system is then utilized to generate two exemplar System-Aware architectures

1. *Blue Perspective* – The prototype decision support system will select a system function to protect based upon the importance placed on protecting that system function by the system design team. First, the prototype system will select the highest ranked (i.e., most important) system function. Next, System-Aware design patterns will be selected to maximize the protection of the chosen system function. Design patterns will be selected until either the costs meet the available budget or all available patterns have been selected. In the former case, design patterns will be selected to maximize the total security effectiveness offered (the sum of the selected combinations of system functions and design patterns) within the user defined budget. This process will be repeated until either all of the identified system functions have been protected by all available System-Aware design patterns or the total cost meets the maximum user defined budget. As discussed in section 3.1.2, it is assumed that the costs and security effectiveness scores for each combination are independent of all other combinations; thus, the total cost and security effectiveness of the candidate architectures can be computed through a simple summation of the individual values.

2. *Red Perspective* – The prototype decision support system will select system functions and corresponding System-Aware design patterns in order to maximize the security effectiveness of the final System-Aware architecture. To do so the prototype decision support system makes the same assumptions as the *Blue Perspective:* that the individually assigned scores for security effectiveness and the costs can be computed from a simple summation in order to derive the values for the candidate architecture. In the event that two or more System-Aware design

patterns consume the same amount of resources and afford the same security effectiveness for different system functions, and only one of those functions can be protected, the importance placed upon protecting the system function by the system design team will be used to determine which of the functions is protected.

For both of these cases, it is recognized that the maximization of the security effectiveness subject to a budget constraint is an instance of the knapsack problem. As a result, the prototype decision support system has been designed to try and take advantage of this fact. First, it is known that the knapsack problem can be solved in pseudo polynomial if all of the weights (i.e., resource costs associated with each combination of system function and design pattern) are non-negative integers. As discussed earlier, the weights are monetary cost estimates. This means that all of the weights are nonnegative. In addition, the maximum budget is an integer value. Finally, the prototype system will round up all of the individual weights to integer values before trying to maximize the security effectiveness. This last step ensures that all of the weights (i.e., costs) are integers. This last step is deemed acceptable as the costs of each of the proposed solutions (i.e., combinations of system functions and design patterns) are considered to be large enough that the change can be safely ignored. Of course, it is still possible that the time required to determine the optimal solution is unacceptable. As a result the prototype systems allows user to use a heuristic algorithm in place of the optimal solution. This algorithm is not guaranteed to find the optimal solution, but it will run in polynomial time.

These candidate architectures are meant to represent two edge cases, thus providing a starting point for user the exploration of the available design

- *Blue Perspective* – Protects the system by protecting system functions according to the system design teams evaluations
- *Red Perspective* – Protects the system by maximizing the security effectiveness of the final architecture based upon the evaluations of the cyber attack assessment team

3.1.4.7 Tools to Support User Exploration

After the prototype decision support system has generated the two candidate architectures, the user is then able to adjust those architectures to generate additional candidates. The prototype decision support system offers several tools to support the user in this given task

- Overview and summary statistics describing the candidate architecture generated using the *Blue Perspective* approach. As seen in Figure 18, this includes a list detailing which combinations of system functions and design patterns were selected and summary information about this architecture
  - Deterrence Score: The sum of the deterrence scores of the selected combinations as well as the deterrence score of summing all possible combinations
  - Cost: The sum of the costs of the selected combinations as well as the costs of selecting all combinations

  - Higher Ranked Blue: Represents the number of important system functions that were included in the candidate architecture. A system function is important if its relative ranking is greater than or equal to the maximum ranked function divided by half

  - Lower Ranked Blue: Represents the number of less critical system functions that were included in the candidate architecture. A system function is less critical if its relative ranking is less than half the maximum ranked function

  - Bigger Det Red: Represents the number of design patterns included in the candidate architecture that contributed a significant amount of effective security. A significant amount of effective security is a security effectiveness score greater than or equal to half the maximum possible score. For the prototype decision support system this is a score of 4 or 3

  - Smaller Det Red: Represents the number of design patterns included in the candidate architecture that contributed a small amount to the effective security. A small amount of effective security is a security effectiveness score less than half the maximum possible score. For the prototype decision support system this is a score of 2 or 1

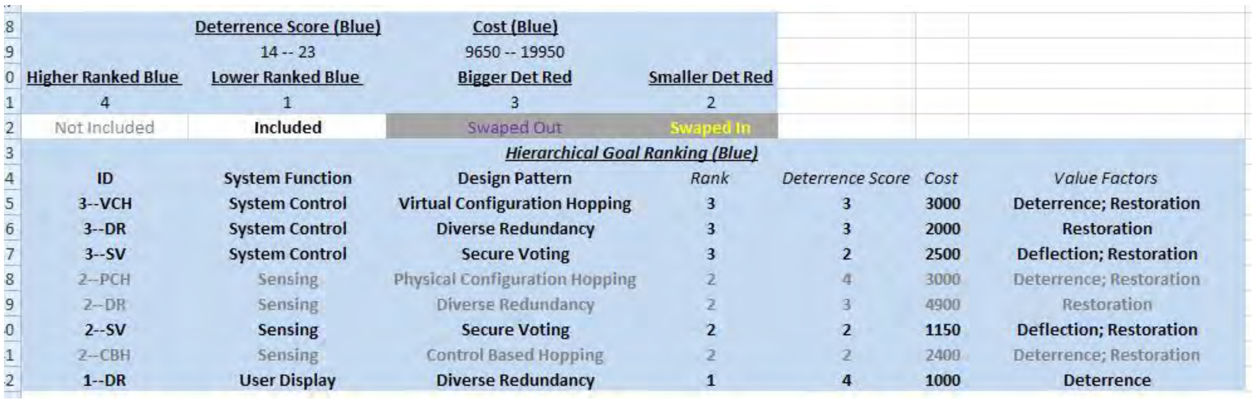
- Overview and summary statistics describing the candidate architecture generated using the *Red Perspective* approach. As seen in, this includes a list detailing which combinations of system functions and design patterns were selected and summary information about this architecture. This information is exactly same as that discussed earlier for the *Blue Perspective*

- Two quad charts representing the summary statistics of the candidate architectures. The first chart, seen in Figure 20, shows all of the combinations including in the candidate architectures (*Blue Perspective* and *Red Perspective*). This information includes the relative importance of the system function being protected and its contribution to the effective security of the candidate architecture. This also includes the amount the combination contributes to the overall costs of architecture (this is represented by the size of the glyph). The second chart, seen in Figure 21, shows the same information as the first; however, the combinations plotted are those NOT included in the candidate architecture

- As discussed in section 3.1.2, the user should be able to create additional architectural candidates by adjusting those generated through more automated means. In the prototype architecture this is done by allowing the user to select a candidate architecture and add or remove combinations of system functions and design patterns. This interface is shown in Figure 22. In addition, the prototype decision support system highlights how these changes affect the selected architecture. These include highlighting which combinations have been added and removed, as well as highlighting any positive or negative changes in the summary statistics. An example of this can be seen in Figure 23

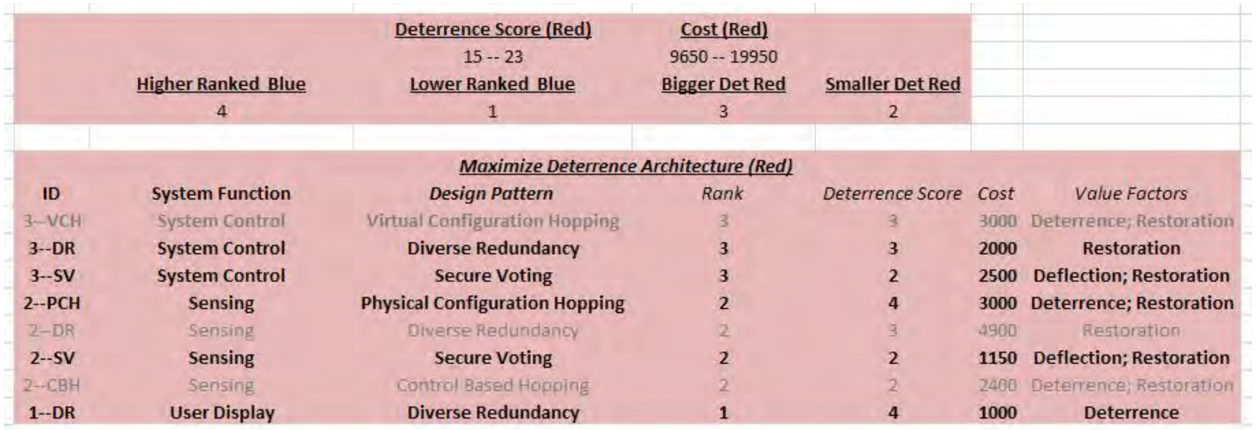
- When the user creates a new architectural candidate, the prototype decision support system allows the user to save that candidate so it can be compared to all other candidate architectures created. The prototype system provides tools to allow for the user to compare candidates in terms of their security effectiveness, costs, and selected combinations of system functions and design patterns in a pair wise manner

- The prototype decision support allows the user to keep a history of all candidate architectures created, the steps (i.e., adjustments to the automatically generated architectures) that were taken to create those candidates, and a history of the reasoning behind those changes



	Deterrence Score (Blue)	Cost (Blue)				
	14 -- 23	9650 -- 19950				
Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red			
4	1	3	2			
Not Included	Included	Swaped Out	Swaped In			
		Hierarchical Goal Ranking (Blue)				
ID	System Function	Design Pattern	Rank	Deterrence Score	Cost	Value Factors
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	System Control	Diverse Redundancy	3	3	2000	Restoration
3--SV	System Control	Secure Voting	3	2	2500	Deflection; Restoration
2--PCH	Sensing	Physical Configuration Hopping	2	4	3000	Deterrence; Restoration
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	Sensing	Secure Voting	2	2	1150	Deflection; Restoration
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	User Display	Diverse Redundancy	1	4	1000	Deterrence

Figure 18. The architecture candidate created automatically using the *Blue Perspective*. The top displays summary information, including deterrence score, total costs, and rough break down of the importance of the system functions protected and the effectiveness of the design patterns chosen to protect those functions. The bottom shows which combination of system functions and design patterns were selected. This instance was generated using the information shown in Figure 15 and Figure 16.



		Deterrence Score (Red)	Cost (Red)			
		15 -- 23	9650 -- 19950			
	Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red		
	4	1	3	2		
		Maximize Deterrence Architecture (Red)				
ID	System Function	Design Pattern	Rank	Deterrence Score	Cost	Value Factors
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	System Control	Diverse Redundancy	3	3	2000	Restoration
3--SV	System Control	Secure Voting	3	2	2500	Deflection; Restoration
2--PCH	Sensing	Physical Configuration Hopping	2	4	3000	Deterrence; Restoration
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	Sensing	Secure Voting	2	2	1150	Deflection; Restoration
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	User Display	Diverse Redundancy	1	4	1000	Deterrence

Figure 19. The architecture candidate created automatically using the *Red Perspective*. The top displays summary information, including deterrence score, total costs, and rough break down of the importance of the system functions protected and the effectiveness of the design patterns chosen to protect those functions. The bottom shows which combination of system functions and design patterns were selected. This instance was generated using the information shown in Figure 15 and Figure 16.

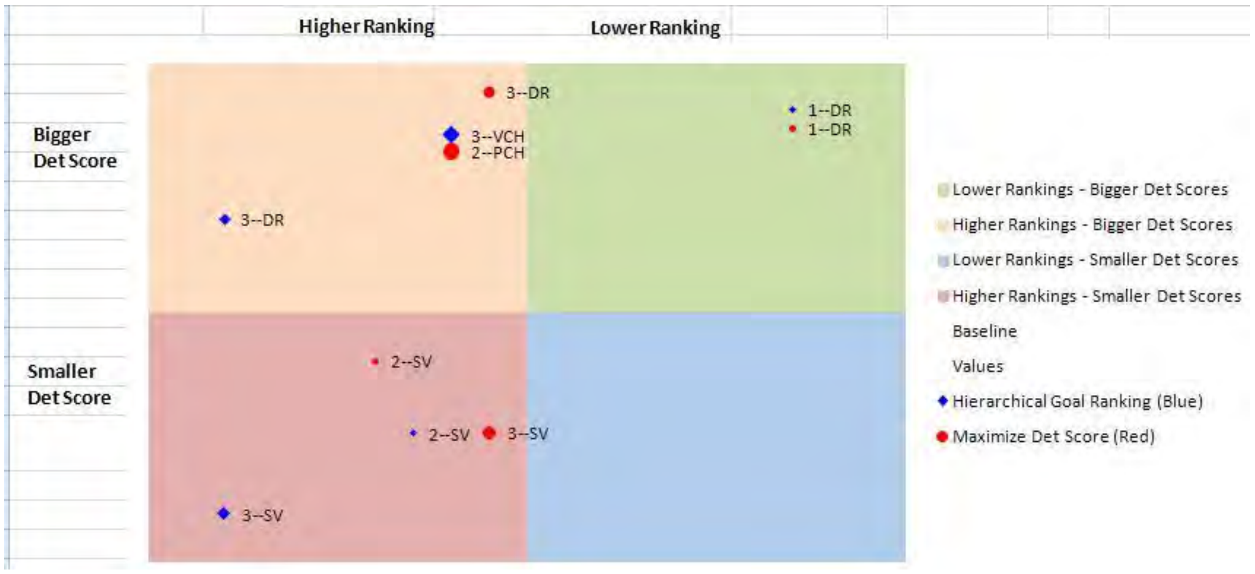


Figure 20. Quad chart displaying all of the selected combinations of system functions and design patterns in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. Those combinations selected as part of the *Blue Perspective* are represented by blue diamonds. Those combinations selected as part of the *Red Perspective* are represented by red circles. The size of the glyph (diamond or circle) represents that combinations relative contribution to the overall cost (bigger glyph represents a larger contribution). This instance represents the candidate architecture shown in Figure 18.

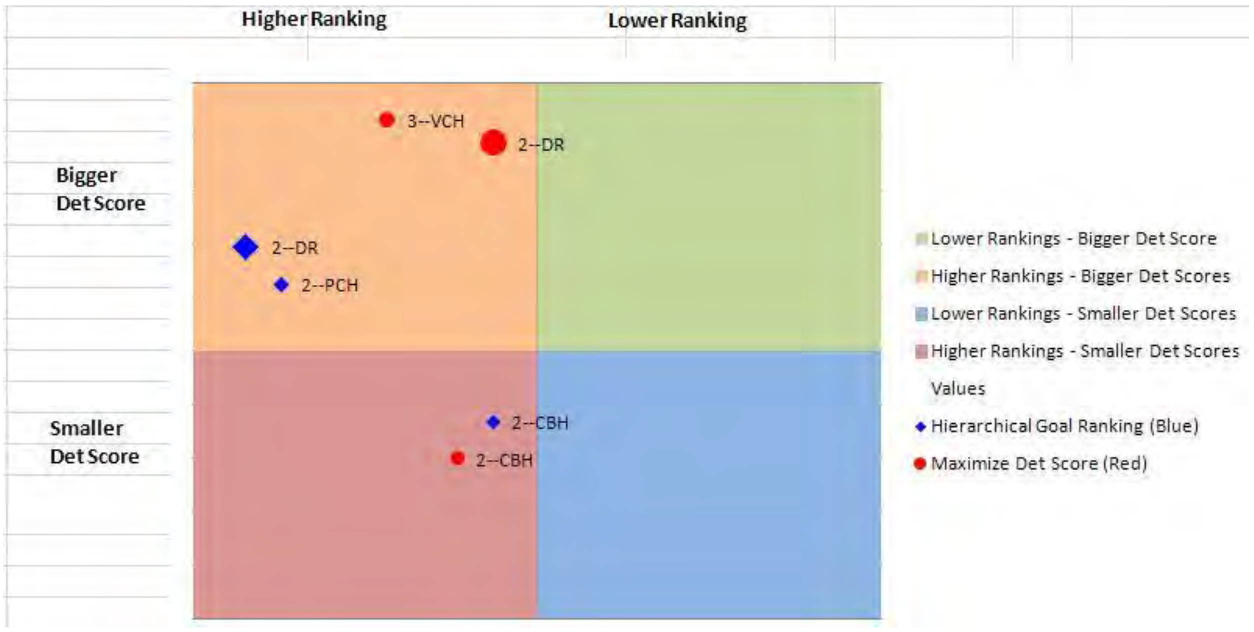


Figure 21. Quad chart displaying all of the combinations of system functions and design patterns not included in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. Those combinations not selected as part of the *Blue Perspective* are represented by blue diamonds. Those combinations not selected as part of the *Red*

Perspective **are represented by red circles. The size of the glyph (diamond or circle) represents that combinations costs compared to the costs of the other combinations not included in that perspective. This instance represents the candidate architecture shown in Figure 19.**

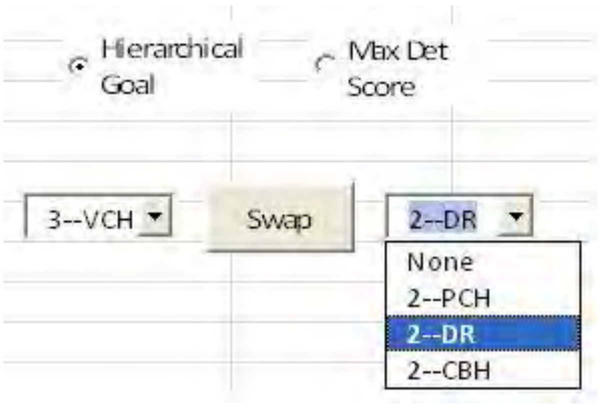
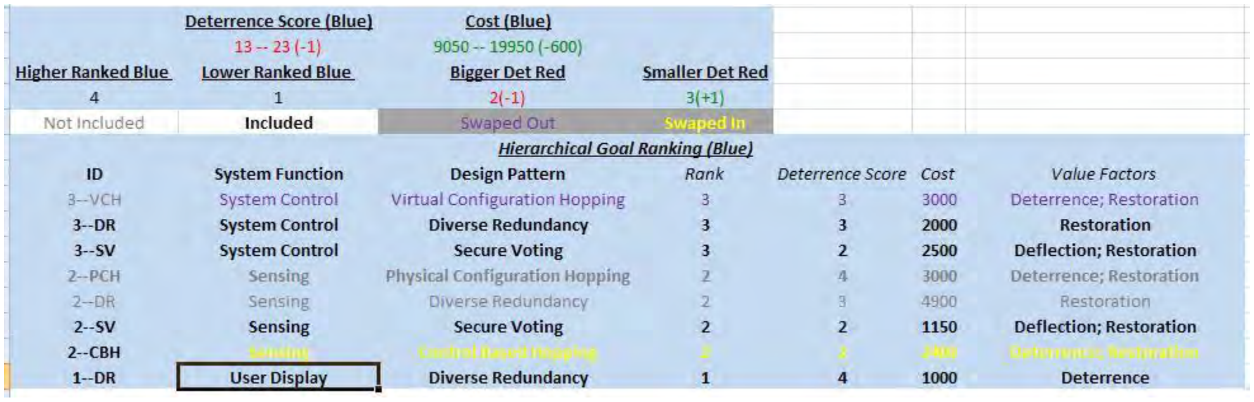


Figure 22. Set of tools the user can use to adjust the candidate architectures generated by the prototype decision support system (*Blue Perspective* and the *Red Perspective*). This includes the ability to select an architecture to modify—Hierarchical Goal (*Blue Perspective*) or Max Det Score (*Red Perspective*)—a selected combination of system function and design pattern to remove (left of the Swap button), and a combination not included in the candidate architecture to add (right of the swap button). This instance represents the information show in Figure 18.



	Deterrence Score (Blue)	Cost (Blue)	
	13 -- 23 (-1)	9050 -- 19950 (-600)	
Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red
4	1	2(-1)	3(+1)
Not Included	Included	Swaped Out	Swaped In

Hierarchical Goal Ranking (Blue)

ID	System Function	Design Pattern	*Rank*	*Deterrence Score*	*Cost*	*Value Factors*
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	System Control	Diverse Redundancy	3	3	2000	Restoration
3--SV	System Control	Secure Voting	3	2	2500	Deflection; Restoration
2--PCH	Sensing	Physical Configuration Hopping	2	4	3000	Deterrence; Restoration
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	Sensing	Secure Voting	2	2	1150	Deflection; Restoration
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	User Display	Diverse Redundancy	1	4	1000	Deterrence

Figure 23. Illustrates how the prototype decision support system supports the user as they adjust the candidate architectures into new architectures. In this instance the combination of system function and design pattern with ID 3—VCH was removed and the combination with ID 2—CBH was added. The changes to the effectiveness and costs of the architectures are show in summary statistics. Furthermore, these changes have been highlighted to indicate (potentially) positive (green) or negative (red) changes.

3.1.4.8 Additional Use Case – The Navy 10th Fleet

During the course of developing the prototype decision support system, it was utilized as part of workshop with the Navy 10th Fleet. This afforded the opportunity to learn more about the benefits and limitations of both the prototype system and the Cybersecurity Architectural Selection and Assessment approach generally. However, before discussing the lessons learned several important limitations should be noted. First, the entire process was condensed into a short 1.5 hour window. This means that the process had to be simplified in order to fit into the allotted time. This led to many of the steps being streamlined so as to create fewer categories for ranking. Second, due to the short window and small number of participants, separate teams could not be created for each of the steps as outlined in section 3.1.2. Instead, all of the steps were performed by having all participants serve as the members for each team.

Despite the limitations of the meeting's setting, the proposed decision support tool was able to provide the group with a useful way of designing and selecting a System-Aware architecture. This included the ability to aid in the identification of important system functions as well in the selection—and creation—of System-Aware design patterns to protect those system functions. In addition, several additional areas for improvement were suggested. First, it is important that the steps be well structured and all participants have some training with process. For example, those participants that were more familiar with the tool and its usage were more readily able to participate in the exercise. However, those participants that were engaging in the activity for the first time were unsure how to provide the rankings or whether to begin with ranking the relative importance of system functions or the potential deterrence afforded by a given System-Aware solution. Second, having more technical experts (i.e., the Red, Blue, and Green teams) available to provide answer decision makers questions would aid in the scoring process. Third, the prototype system currently assumes that all system functions are prioritized only according to the possible consequences that could result from a cyber-attack; however, initial usage suggest that the prioritization of a system's functions is, in fact, a combination of multiple factors. For example, during the discussion all system functions were found to classified into three broad categories, (1) functions related to the system's operations, (2) functions used to carry out the system's specific mission objectives, and (3) functions related to the system operators ability to control the system. Some members considered those functions related to the performance of the system as most critical: believing that these functions could be used to cause catastrophic damage to the entire system and compromise the mission. Other members disagreed with this assessment. They believed that such attacks could be easily identified and deflected by other means—such as operator intervention—and, furthermore, may degrade the system's functionality but not prevent it from accomplishing its mission. Finally, all members noted that those system functions that would be built from more stable components should be ranked higher; i.e., some system functions may be built using components that will be in service for years, while others may be built from components that will be upgraded frequently. Those system functions that will be upgraded frequently should receive a lower priority as the constant upgrading would provide a certain degree of protection by possibly deterring an adversary.

Currently the prototype system only provides the cyber attack assessment (red) team with the ability to assign an integer value of one to four to the security afforded by each system function design pattern solution. During the discussion, it was suggested that a larger range of values be available to allow the red team to more accurately assess the security offered by a given solution.

3.1.5 Model-based Cyber Assessment

As described in Sections 3.1.1 to 3.1.4, the scoring and influence diagram approaches to architectural selection were used successfully in the context of the UAV system. These applications made it clear that the methods were less than ideal in that they exhibited the following characteristics:

- A lack of associated model semantics giving clear and repeatable meaning to the model elements.
- Manual construction that proved to be both tedious and time consuming.
- Absence of integration between the system (defender) and attacker perspectives.
- Limited support for understanding tradeoffs between cost, system impact, security, and other objectives.

More generally, the Phase 1 methods do not support application to complex systems or systems-of-systems or use in agile planning environments. The Phase 2 activity in architectural selection focused on investigating the hypothesis is that a scalable and agile approach to the mission-focused architectural selection problem can be found by making use of two structured modeling approaches:

- **Systems models** using model-based systems engineering tools and languages such as SysML , which provides a mechanism for detailed description of of system structure, functions, and information flows in a searchable data format.
- **Attack models** using attack tree tools, such as SecurItree, which provides a paradigm (similar to fault trees) that can be used to describe vulnerabilities to cyber attack and that can be filtered to account for the particular capabilities or characteristics of potential adversaries.

Research, described in Part 4 of this reports, was directed in the following thrusts:

- Refinement of the architectural **selection methodology** that is an iterative process in which defender, attacker, and procurement viewpoints are alternately considered in the choice of which defenses to employ where.
- Creation of **model integration concepts** that support a unified system and attack model.

See Part 4 of this report for more details.

3.1.6 Future Activity in Systems-aware Architectural Selection and Assessment

The Phase 2 activity led to substantial progress in the development of model-based tools and techniques to support Systems-aware Architectural Selection and Assessment. However the hypothesis that these tools yield a truly scalable approach remains largely untested. In our opinion, future efforts should focus on the following items:

- Continued development of integration between systems and attack tree models.
- Extension of the six-step methodology to support application in hierarchical and systems-of-systems settings.

Application of the methods in hierarchical and systems-of-systems case studies concepts to assess the overall effort involved in the selection of the defensive architecture and the degree to which the concept can support an agile environment through the reuse or combination of architectures.

3.2 Attack Development

3.2.1 Parameter-Based System Attack

UAV autopilot systems are designed to be reusable across a diverse set of aircraft configurations and support a variety of mission scenarios. As a result, many of the variables that govern the control algorithms for a given flight are parameterized:

- Maximum and current fuel capacity.
- Maximum allowable pitch, yaw, and turning radius.
- Maximum altitude the aircraft can safely operate.
- Flight plan for a given mission.

However, while these parameters allow a given autopilot to fulfill a large number of missions, they also provide a potential attack vector that a malicious adversary could use to damage an UAV or compromise its ability to carry out its mission objectives. For example, as discussed in section 3.2.3, UAVs are primarily used as platforms for providing ISR. An adversary could use a parameter-based attack to neutralize a UAVs ability to carry out its surveillance operation through the usage of an embedded Trojan horse that would be capable of disrupting the UAV’s ability to gather surveillance in key regions; i.e., the Trojan horse would alter the UAV's flight plan when the UAV entered certain geographic regions.

This section will outline one potential parameter-based attack vector against a UAV autopilot. This attack will be in the form of a Trojan horse designed to modify the UAV's flight plan stored in the autopilot system. The adversary will leverage the fact that the flight plan is stored in the autopilot system as a series of waypoints (i.e., destinations) that the aircraft will fly between. When the aircraft enters a key geographic region, an embedded Trojan horse will automatically divert the aircraft to another waypoint in the flight plan.

For the UAV navigation system attack example, the red team constructed three trees for three different attack types; each attack type potentially possessing a different value to a possible attacker:

1. A *minor trajectory change* where the adversary makes a minor change to the waypoints in order to cause the platform to deviate slightly from the flight path. The intent of this attack is to prevent the UAV from operating in designated regions. This attack assumes that the deviation’s magnitude and duration are small enough to go unnoticed by the aircraft operator.

2. A *major trajectory change* where the adversary drastically alters the flight trajectory in order to cause the platform to lose control and crash into the ground. This attack assumes that the flight trajectory alterations occur too quickly for the pilot to prevent the aircraft from losing control and crashing.

3. A *concealed major trajectory change* where the adversary drastically alters the flight trajectory in order to reroute the UAV to an alternate destination. This attack assumes that the trajectory change on its own will be noticed and can be prevented by an operator taking appropriate action(s). As a result, this attack assumes that the adversary will take action to conceal or mask the major trajectory change in order to prevent the operator from taking any actions that might thwart the attack.

The *Concealed Major Trajectory Change* (option 3) tree was selected for the analysis moving forward into testing for two reasons. First of all, structurally, all three trees are similar in regards to how the exploit is realized. This results in the *Concealed Major Trajectory Change* tree including the nodes of the other two as well as the nodes representing actions to lower the detectability of the attack. Second, the value gained from the *Concealed Major Trajectory Change* attack was most in line with the expected preferences of the adversary profiles the project team was most concerned with. For demonstration purposes in this project, the project team for this application chose implementation of the parameter assurance for the waypoint change and it was also enhanced to include protection on both the ground and in the air covering the case where an attacker might launch a coordinated attack to try to hide the effects of a waypoint change from operators on the ground. This is the use-case that was demonstrated in the flight evaluations performed in October 2014.

3.2.1.1 Parameter-Based Attack Implementation Details

There exist multiple insertion points where a malicious adversary could embed the Trojan horse into the Piccolo autopilot in order to divert the aircraft to another waypoint in the flight plan:

1. Directly into the Piccolo autopilot's hardware.
2. Algorithms used to control the aircraft's flight.
3. The Piccolo autopilot provides support to allow up to five external devices to connect serially using the RS-232 protocol. Once connected, these devices are able to passively monitor the flight status of the Piccolo autopilot, extract information from the Piccolo autopilot, and modify the Piccolo autopilot's flight parameters.

The attack outlined in this section assumes that the adversary will utilize option (3):

- All three attack vectors will enable the adversary to alter the flight plan; however, option (3) requires the least modification to the existing Piccolo autopilot.
- Simplifies the reconfiguration of attack parameters; e.g., option (3) makes it simpler to experiment with alternative triggering mechanism used in the attack.
- Attack can be implemented on any platform that can send and receive using the RS-232 protocol, including laptops, desktops, SBCs, etc.

For the Phase 2 implementation of the parameter-based attack, the Trojan horse has been implemented on a Raspberry Pi labeled as *Attack Pi 1* from Figure 5. This Single Board Computer (SBC) is located on the airborne platform and can be executed from the ground on command our automatically based on geographic region. It is connected to the Piccolo autopilot over one of the available serial connections –

COM1. In addition, the Trojan horse is able to be triggered by either entering a specific geographic region, or, to facilitate experimentation, the SBC allows for a malicious user to redirect the aircraft to any waypoint through a simple text based interface.

3.2.1.2 Masking the Parameter-Based Attack to Prevent User Detection

The attack implementation outlined in section 3.2.1.1 would be sufficient to compromise a UAV's capacity to fulfill its designated mission. For example, an adversary would be able to use the embedded Trojan horse to create no-fly zones for the UAV's; enabling them to operate in a given region without the risk of detection. However, changing a UAV's flight plan mid-flight is an action that could be readily recognized by the UAV's operator. As a result, the attack might only be effective for a short duration before it was detected and corrected. In addition, the operator might be able to take actions to salvage the mission; e.g., the operator might call in the assistance of one or more near-by UAVs to take over the mid-mission. Thus, if an adversary desires such an attack to be effective over the course of multiple missions, they will need to take steps to ensure that the attack is not easily detectable.

To reduce the risk that the embedded Trojan horse will be detected, the adversary decides to coordinate the parameter-based attack (i.e., flight-plan alteration) with an attack against the operator display (i.e., PCC). This attack will mask any alterations in the flight plan from the operator display. Specifically, when the Trojan horse embedded into the Piccolo autopilot redirects the UAV to an alternate waypoint the attack against the PCC will display the UAV flying along the previously unaltered flight-plan.

During Phase 2 of the project, as the ground-station portion of the attack, a rudimentary physics engine was developed based on the GPS location of the waypoints in the UAV flight plan. The current position in the flight plan was coupled with the velocity, bearing, direction and type of the flight plan to create falsified data that was inserted into the stream of flight data that determined the location of the aircraft on the maps in the Piccolo Command Center displays. Once the attack on the waypoint was initiated in the autopilot of the UAV, a signal was sent down to the ground that triggered the secondary attack on the operator displays. When the attacks were coordinated, any changes in the flight-plan waypoints were hidden from the operator’s station.

This attack was particularly important in the testing and the safety of our flight tests. To maintain the safe operations of the aircraft, it was decided that it was necessary to have a second PCC workstation as a backup in to the PCC that would be attacked during the flight tests. In case of loss of control, the second workstation and the second operator could both see the real picture of what the aircraft was doing at that time and have the capability to take over control of the aircraft if needed. In the tests, this turned out to also offer an interesting test for the System-Aware protections, which we will describe in the Special Note in section 3.2.1.4.

Figure 24 shows how the attack platform used to attack the operator display is integrated into the HiL emulation environment. In this instance, the attack is hosted on a SBC that intercepts all communication between the operator display (PCC) and the ground station. As was the case for the Trojan horse embedded onto the UAV, this configuration affords an easily reconfigurable attack platform to facilitate experimentation while minimizing its impact on existing systems. It is noted that this is not the only available point for insertion. For example, the display masking attack could be embedded into the PCC itself.

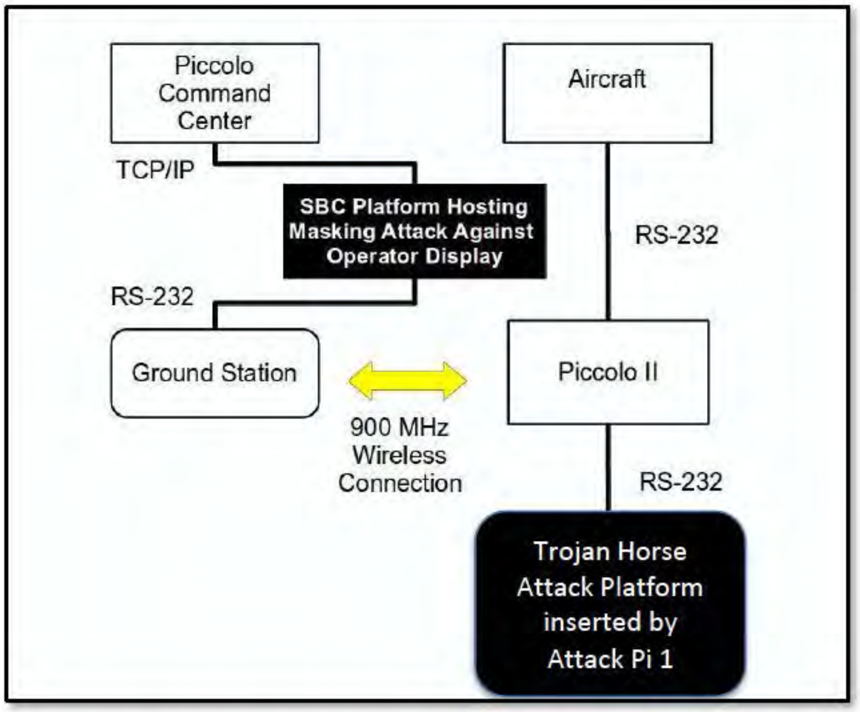

Piccolo Command Center
Aircraft
TCP/IP
SBC Platform Hosting Masking Attack Against Operator Display
RS-232
RS-232
Ground Station
Piccolo II
900 MHz Wireless Connection
RS-232
Trojan Horse Attack Platform inserted by Attack Pi 1

Figure 24 - Embedded attack platform for masking the operator display on a SBC.

3.2.1.3 Implementation of the Parameter-Based Attack by Compromising the PCC

As described in section 3.2.1.2, it is possible for the adversary to initiate a clandestine parameter-based attack against the Piccolo autopilot. However, such an attack requires two embedded attack platforms working in a coordinated fashion. Furthermore, each of these platforms has to be embedded into two distinct subsystems. This section explores an alternative attack vector requiring the adversary to compromise only one subsystem.

Figure 25 shows the configuration for the HiL emulation environment that initiates a ground-based version of the same parameter-based attack as described section 3.2.1.2. For this configuration the embedded Trojan horse responsible for initiating the parameter-based attack is embedded directly into the PCC using its plugin capabilities. This attack has the advantage of only requiring the adversary to compromise the ground-based operator display; however, it is also a potentially easier attack to purge. As the attack is not embedded with the Piccolo autopilot, it can be purged mid-flight by simply swapping in an uncompromised PCC to control the aircraft. This is identical to the actual Phase 2 flight hardware which utilized the same functionality to attack the ground components.

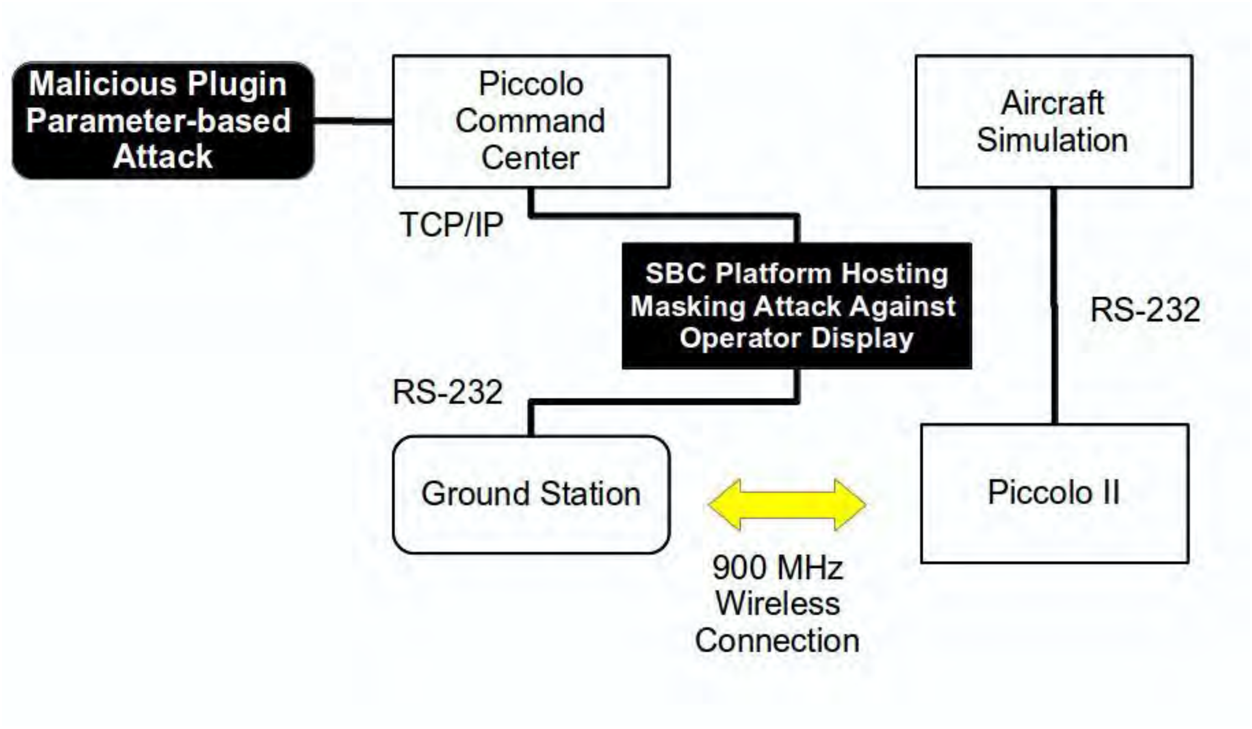

Malicious Plugin Parameter-based Attack
Piccolo Command Center
Aircraft Simulation
TCP/IP
SBC Platform Hosting Masking Attack Against Operator Display
RS-232
RS-232
Ground Station
Piccolo II
900 MHz Wireless Connection

Figure 25. Piccolo II HiL configuration with Embedded Attack Platform to dynamically direct the UAV to a specified waypoint during flight while masking the change from operator display. Attack is a plugin embedded into the PCC. Embedded attack platform for masking the operator display is a SBC that sits between the connection from the PCC and the ground station.

3.2.1.4 Flight Demonstration - The Waypoint Attack

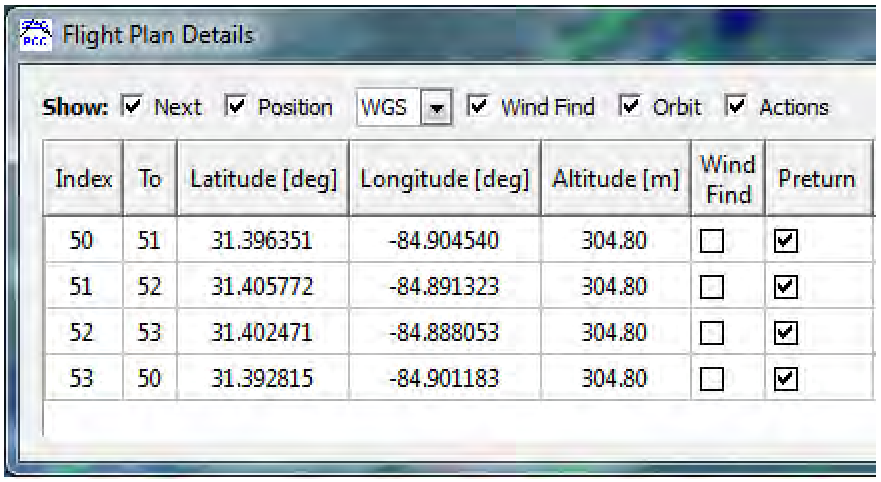
The waypoint attack changes the waypoints in the autopilot's flight plan causing it to flight a different trajectory from the one intended by the operator. As shown in Figure 26 the aircraft's commanded flight plan was one of two rectangular patterns aligned with the runway. The other rectangular pattern was used as the set of waypoints for the attack flight plan. The coordinates of the waypoints are provided in Table 1 or Table 2.

The flight director initiated three flight passes in order to test the use of the Sentinel to monitor and to detect system behavior for possible cyber attacks. During the first evaluation flight pass, the operator initiated a normal waypoint change. The Sentinel successfully noted the change in plan, verified that the change did come from the operator, and indicated that a normal change had occurred in the Cyber Commander Interface. During the second evaluation flight pass, an attack was triggered that injected a false waypoint change into the flight plan. The Sentinel successfully noted the change, did not find an associated operator system change to the plan, and indicated a possible cyber attack to the Cyber Commander. The interface offered the appropriate corrective actions to the operator which included restoring the original flight plan, flying the aircraft back to its home location, and flying to another waypoint or flight plan of the operator's choice. The demonstration scenario called for a change back to the original plan.

For the third flight plan pass, the attack on both the airborne and the ground-station components were triggered in order to both change the waypoint plan and hide the result from the operators. During the initial portion of this part of the demonstration, the aircraft was commanded by the operator to fly the rectangular pattern consisting the waypoints in Table 1. With the defense disabled the flight test director initiated the waypoint attack and the aircraft diverted to the other rectangular pattern made up of the waypoints in Table 2. Unfortunately, the masking portion of the attack did not work properly. When the masking attack was engaged, the aircraft's position froze on the 900 MHz ground control station display. The aircraft's true location could still be seen on the 450 MHz ground control station display. During the next pass through the pattern the defense against the waypoint attack was enabled. This time, when the attack was initiated, the Sentinel alerted the operator to the change in the flight plan. We believe that this communications problem may have been caused by a disagreement between the physics engine that enabled the masking attack to the PCC and the information being sent from the aircraft to the ground. This had been tested thoroughly in the SiL and HiL testing environments. However, more testing will be needed to find the problem from the airborne tests.

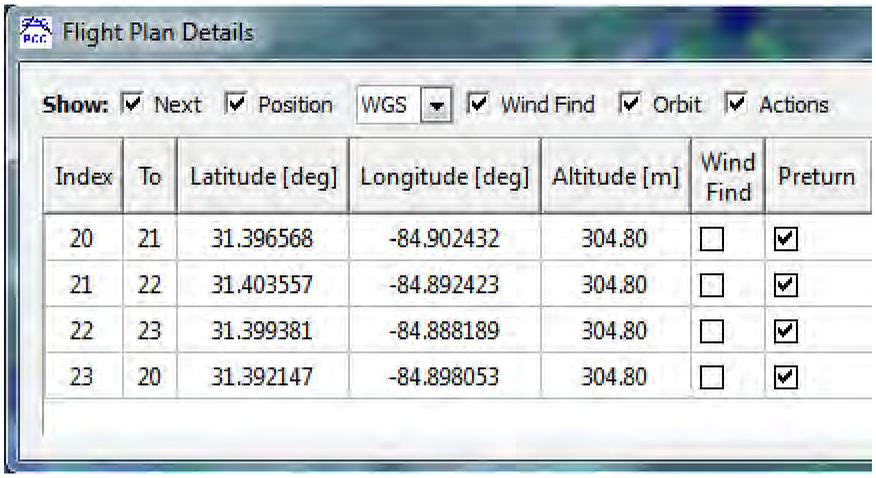
Special Note: An interesting and most important result came from this communications failure. Because the system froze on the 900 MHz PCC when the secondary attack was initiated on the main operator's center, the operator of the backup, 450 Mhz ground control system was forced to make changes to the flight plan to keep the aircraft flying in the correct flight plan. During the setup of the tests, the Sentinel was not made aware of the presence of the 2nd flight control station. So, when the operator initiated the change on the unknown station, the Sentinel detected the change as an unverified change and alerted the Cyber Command station that a cyber attack was underway. This attack is very similar to a spoofing attack where a false operator station would attempt to take over flight operations for a UAV. This was a critical point in the tests where the Sentinel worked as designed to protect the flight plan in an unplanned scenario and was a defining moment for the tests. The Sentinel was doing what it was designed to do, protect the system functions and offered corrective actions to the operator.

Table 1. Waypoints on North West side of runway



Index	To	Latitude [deg]	Longitude [deg]	Altitude [m]	Wind Find	Preturn
50	51	31.396351	-84.904540	304.80	☐	✓
51	52	31.405772	-84.891323	304.80	☐	✓
52	53	31.402471	-84.888053	304.80	☐	✓
53	50	31.392815	-84.901183	304.80	☐	✓

Table 2. Waypoints centered over runway



Flight Plan Details

Show: ☑ Next ☑ Position WGS ☑ Wind Find ☑ Orbit ☑ Actions

Index	To	Latitude [deg]	Longitude [deg]	Altitude [m]	Wind Find	Preturn
20	21	31.396568	-84.902432	304.80	☐	☑
21	22	31.403557	-84.892423	304.80	☐	☑
22	23	31.399381	-84.888189	304.80	☐	☑
23	20	31.392147	-84.898053	304.80	☐	☑

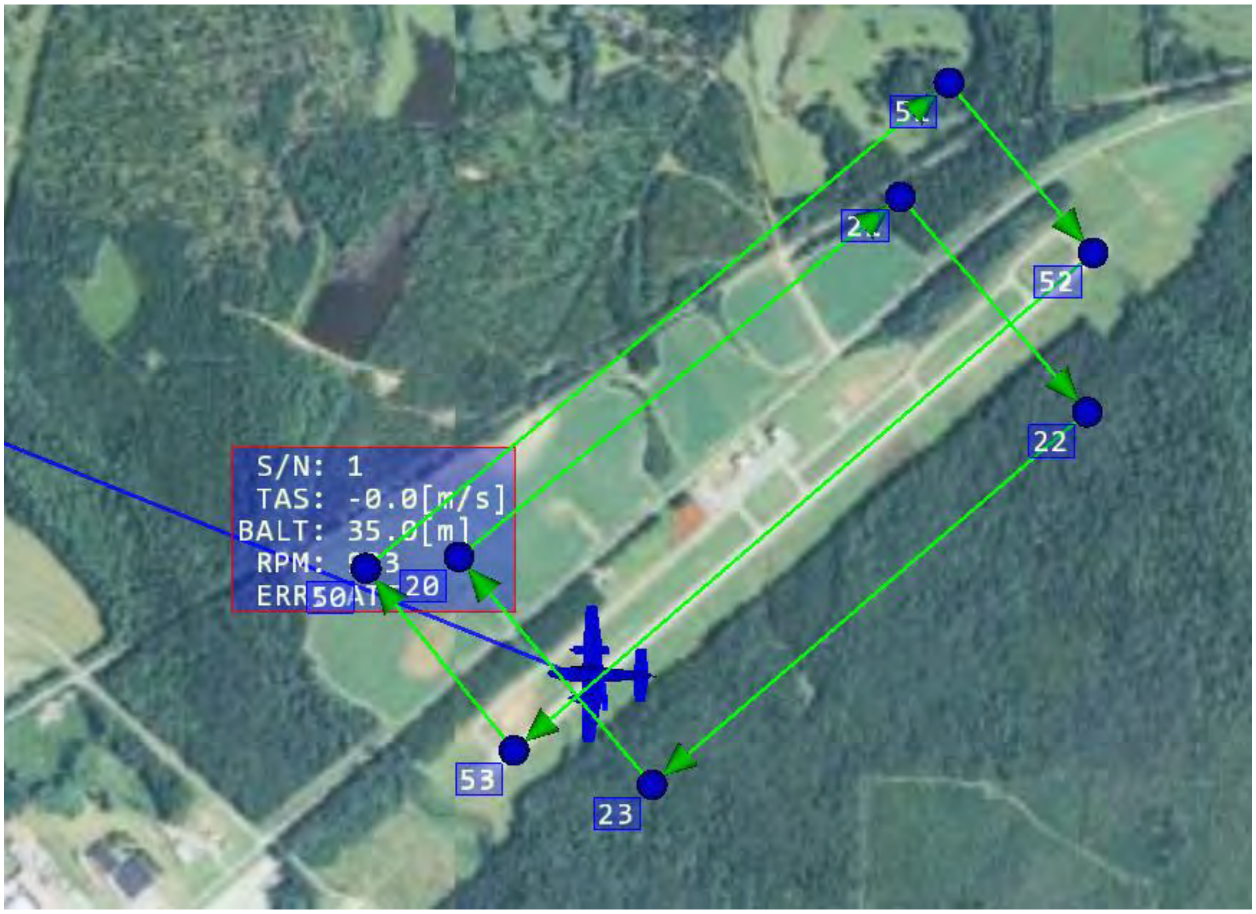


Figure 26. Waypoint Attack Flight Plans

3.2.2 GPS System Attacks

3.2.2.1 Applying the Relational System Aware Architectural Assessment Methodology to GPS

Step 1

The process begins by identifying the critical functions of the gimbal system and defining the variables and influence relationships among those functions. Step one is to be performed by the blue team and is intended to outline the expected functionality of the system with minimal defensive strategies implemented. At this point, a system influence relational diagram is constructed using DAG notation as described previously.

Figure 27 shows the influence diagram for the gimbal camera pointing system. As seen in the diagram, the subsystems of interest are the camera control processor, GPS, and the sensor and effector group. The camera control processor executes SW to implement functions such as Pan-Zoom-Tilt, auto-tracking, point of interest tracking, etc. The GPS receiver provides the necessary geo-reference data to the control processor and camera to locate and track objects of interest. The sensor and effector group provides motion-stabilization to the mounted camera during flight. The camera control processor also is a downstream device to the GPS receiver; thus, it is also a possible host for embedded malware to alter GPS measurements as they are streamed into the control processor. The GPS receiver, gimbal sensors and effectors, and gimbal control processor are the three systems that most influence the success or failure of a given surveillance mission. Of these three, the GPS receiver is of particular interest because GPS measurements greatly influence the geo-referencing of the image data from the camera.

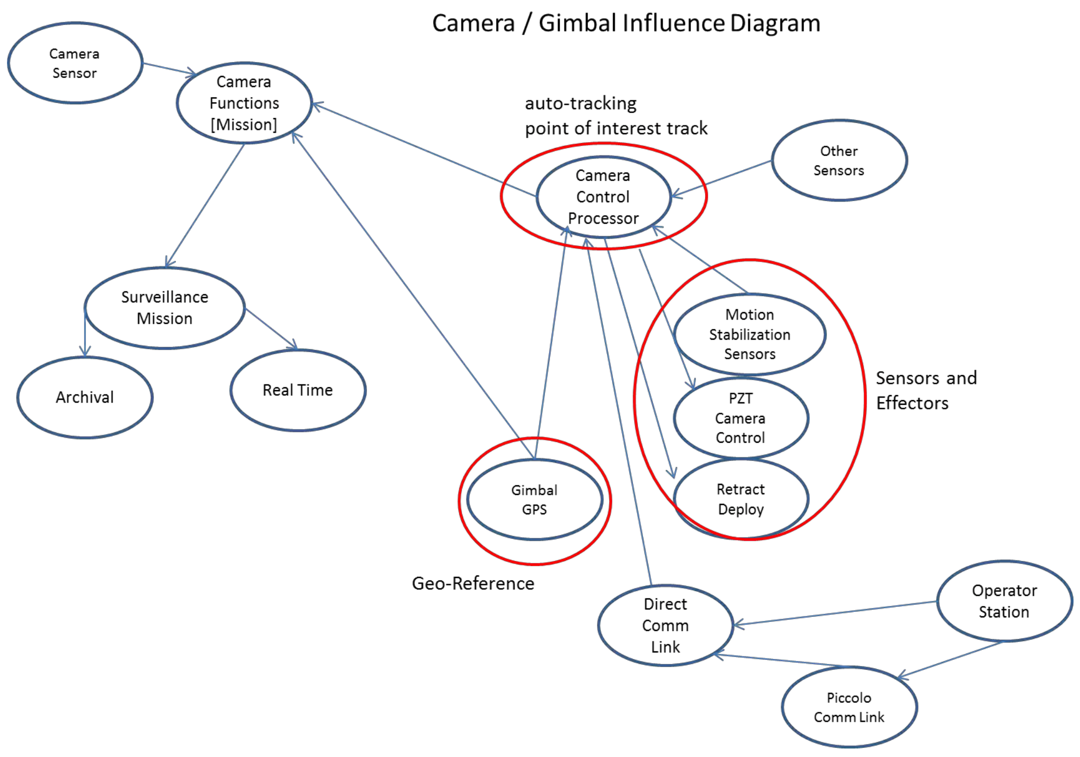


Figure 27. Camera gimbal influence diagram.

Step 2

In step two, the red team is tasked with constructing an attack tree for the specific system function considered in step one. In this case, it is the attacks on GPS metadata. Metadata is all of the recorded by the gimbal and the UAV to provide a complete solution to geo-referencing the surveillance information collected by the UAV. This typically includes gimbal-pointing angles, PZT of the camera, UAV attitude data, GPS data, etc. By looking at the system from the perspective of an adversary, attack trees can be utilized to understand the possible paths an attacker could take to exploit a specific feature of the system. The attack tree in Figure 28 represents the possible vulnerabilities an adversary could exploit to alter GPS measurements in the metadata stream. The attack tree is organized into four viable attacks categories:

Supply Chain Attack on GPS Receiver: A Trojan embedded into the firmware of the GPS receiver.

Down-stream GPS Malware Attack: An attack on a downstream device that is receiving GPS data. The GPS data it receives is manipulated before it is used by the device.

Manipulated GPS Firmware Attack: An attack injected during system integration. In this scenario, updated system patches or firmware for the GPS receiver is altered prior to being loaded onto the GPS receiver.

External GPS Attack (Spoofing): Spoofing one or more GPS signals external to the UAV from a phase-coherent spoofing device that causes the GPS receiver to falsely lock onto the spoofed signals.

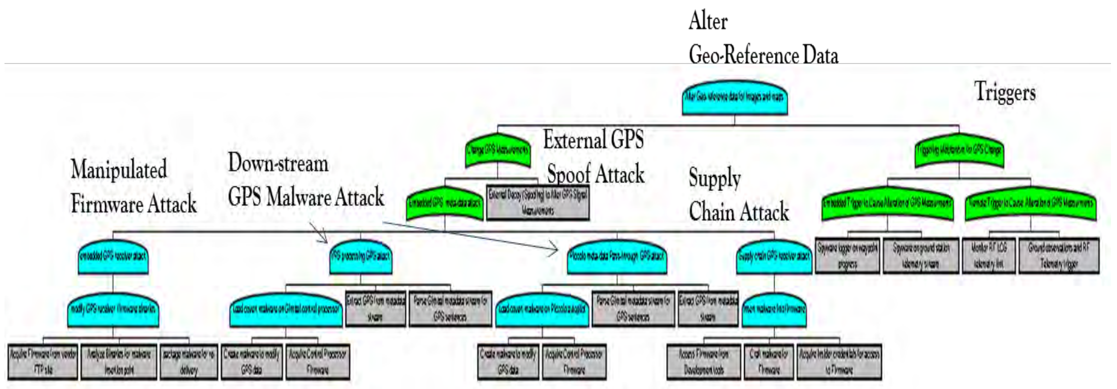


Figure 28. Attack tree for GPS attacks to alter georeference data.

In Figure 28 the attacks are organized from left to right based on difficulty of executing the attack. The first attack is the *manipulated firmware* attack. GPS devices are pervasive in consumer products; as such, the suppliers of GPS often provide open API's and a variety of firmware packages to suit the needs of a diverse customer base. The firmware packages offered are usually placed on a FTP server (File

Transfer Protocol) for customers to retrieve. This leaves them open to skilled adversaries who can retrieve the GPS firmware by either masquerading as a legitimate client or simply crack the FTP site. Once the firmware binaries are downloaded, reverse engineering methods and tools can be applied to the firmware to deduce its functionality and operations. After analysis of the firmware is complete, suitable locations in the firmware are selected for inserting malware to alter GPS position calculations based on trigger. The compromised malware is then delivered to a specific target system integrator or user of the UAV. There are a number of delivery mechanisms that can be used to fool the vendors into inheriting the compromised firmware. One such method employs special probe detectors in the vendor's servers that detect a request to the firmware FTP site by the vendor personnel. The probe detectors will allow the download request to proceed, but it will swap and replace the authentic firmware with the compromised the compromised firmware unbeknownst to the users.

The second attack is an indirect attack on GPS measurements. That is, the systems that use or process GPS sensor data are compromised in such a way that the GPS data is altered before they can use it. This type of attack exploits vulnerabilities in the system dataflow protocol and development tools for the systems that use the GPS data. In this attack, the communication API's of the system that define how data is formatted, where certain data is identifiable by header packets, and protocol for communications is exploited for intercepting and modifying GPS data before it is used by the downstream device. Typically, this attack requires a man-in-the-middle attack posture. In our case the GPS device and Gimbal Control Processor are directly connected together via a serial link. Thus, the man-in-the-middle exploit will reside on the input of the control processor as part of the input processing software or protocol conversion software. In either case, it requires some form of malware to be loaded onto the control processor. The malware will alter the GPS database on a trigger.

The next type of attack is a supply chain attack. In this case, an insider in the GPS vendor company is colluding with an external agent to place stealthy malware deep into the GPS firmware. The insider must have access and authorization to configure the firmware on the SW development tools, hide the changes from test engineers, and be skilled enough to craft or insert the malware in the right place. This type of attack requires the coordination of many complex activities involving human intelligence, skilled adversaries to work with an insider, and circumventing product security measures. This attack is the most difficult, only possible in the realm of highly developed adversaries, but can extremely effective.

The last type of attack is external GPS signal spoofing. All of these exploits are externally executed through a special RF spoofing device (phase-coherent signal synthesizers)—a device that simultaneously receives and transmits civil GPS signals. This type of attack causes the GPS receiver to falsely lock onto a fake GPS signal that is used to provide false updates to the UAV systems. We provide this type of attack in the tree as a measure of completeness, we do not intend to investigate countermeasures to this type of attack as it out of scope of this project, and has been widely researched by others.

Triggers for GPS system attack are considered as well. Typically, an adversary who has embedded a GPS exploit in the gimbal system would want to coordinate the attack with some form of trigger. An example of a trigger could be when the gimbal camera system is deployed activate the attack, or when the GPS stream data indicates XYZ latitude and longitude coordinates activate the attack. These triggers can be embedded internally with malware or perhaps triggered externally. For an external trigger to work, the adversary would have to gain access to the RF telemetry channel that is used to communicate to the ground station. This could possibly be accomplished by spoofing the telemetry channel on duplicate but a higher powered transmitter of the same type as the ground station. Alternatively, the ground station software could be compromised in such a way as to stealthily upload trigger commands to the gimbal system.

Step 3

In step three, the red and blue team develops a set of variables that can be used to assess the difficulty of a particular attack action. These variables are called *behavioral indicators* and can include, but are certainly not limited to, resources such as technical ability, time, manpower, money, equipment, facilities, presence of an insider, and access to system design information. These variables are used to make two separate types of judgments: leaf node assessments and adversary profile construction. The adversary profile is the characterization of an attack agent. In our work, these are nation states, cyber-criminal groups, terrorists groups, and rogue agents. Leaf node assessments are directed with respect to a particular adversary group. Annotating the leaf nodes with a graded five-point scale from low to high provides the basis for pruning the attack tree to select attacks that are desirable to attackers. An example of a pruned tree is in Figure 29, where the supply chain attack has been pruned due to the relative difficulty of the attack for the rogue agent to perform. Pruning is always done with respect to a specific threat agent profile; as such, the supply chain attack would not be pruned for the nation state threat agent because it is within their capability to conduct such a complex attack.

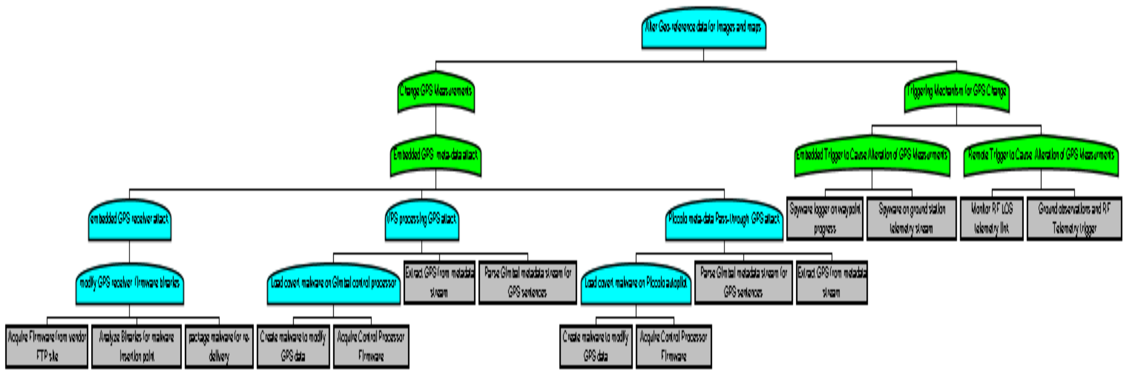

Alter Geo-reference data for Images and maps
Change GPS Measurements
Triggering Mechanism for GPS Change
Embedded GPS meta-data attack
Embedded Trigger to Cause Alteration of GPS Measurments
Remote Trigger to Cause Alteration of GPS Measurments
embedded GPS receiver attack
VPS processing GPS attack
Piccolo meta-data Pass-through GPS attack
Spyware logger on waypoint progress
Spyware on ground station telemetry stream
Monitor RF LOS telemetry link
Ground observations and RF Telemetry trigger
modify GPS receiver firmware binaries
Load covert malware on Gimbal control processor
Extract GPS from metadata stream
Parse Gimbal metadata stream for GPS sentences
Load covert malware on Piccolo autopilot
Parse Gimbal metadata stream for GPS sentences
Extract GPS from metadata stream
Acquire Firmware from vendor FTP site
Analyze Binaries for malware Insertion point
package malware for re-delivery
Create malware to modify GPS data
Acquire Control Processor Firmware
Create malware to modify GPS data
Acquire Control Processor Firmware

Figure 29. Pruned Attack Tree.

Based on the analysis of step three, all attacks are within the capability of the nation state threat actors. Rogue agents, cyber-criminals groups, and terrorists groups can execute manipulated firmware attacks and downstream GPS attacks. Cyber-criminal groups can additionally execute GPS spoofing attacks. For the remaining analysis in steps four through six we focus on nation state actors and evaluate cyber defenses for the gimbal system. Based on the attack tree analysis the most desirable attacks for a nation state actor are ranked in the following order:

1. Manipulated Firmware attacks
2. Downstream GPS malware attack
3. External Spoofing
4. Insider supply chain attack

Steps 4-6

Step three identified attack scenarios and actions that an adversary would need to take to successfully execute an attack and those that are most attractive to a particular adversary. Based on the ranked attack scenarios the blue team can determine which cyber defensive actions may be appropriate to provide strong asymmetry against the attack scenarios. Referring to Table 3, the design patterns are assessed with respect to the four attack scenarios. Column 1 is the attack scenario provided by step three. Column 2 is the selected design pattern to defend against the attack. Column 3 is the implementation cost of the design pattern. Column 4 is the collateral system impacts of the design pattern; i.e., how the design pattern negatively impacts the performance of the system. The design patterns evaluated for the GPS gimbal attacks are diverse redundancy of GPS modules and verifiable voting of the diverse GPS module measurements. Diversity of GPS modules provides defense against supply chain attacks. Redundancy of GPS modules provides defense against directed attacks on a specific GPS modules firmware. Redundancy and consistency checks are required for detecting a downstream GPS attack. Recall in a downstream attack, the source GPS module is not compromised but the downstream components that use GPS measurements are corrupted. In this case, the downstream devices need a consistency check on their GPS data, and this is accomplished by feeding back their GPS measurements to the Sentinel for checking against the redundant measurements. In the presence of an attack, the downstream devices received measurements that would be different from the ensemble of redundant GPS modules. GPS spoofing can be detected by a number a methods that are available in the open literature. The Tippenhauer is one such method. The basic idea of the Tippenhauer countermeasure is the following: four GPS receivers are placed on the UAV separated by at least 4 meters. The distances between the GPS is accurately surveyed and known. If the GPS receivers can exchange their individual GPS surveyed locations, they can check if their calculated locations preserve their physical formation (within certain error bounds). In the case that the calculated GPS locations do not match the known formation, an attack must be suspected and there should be a warning message. This defense requires additional GPS receivers (beyond what is needed for UAV operations) to be placed on the UAV at maximal separation points of the vehicle, such as the nose, tail and wingtips.

Attack Type	Design Pattern	Implementation Cost	Collateral System Impacts
Embedded GPS Receiver Attack (Supply Chain Attack)	Diverse Redundancy of GPS Modules and Verifiable Voting	Low-Med	Low-Med
Down-stream GPS Malware Manipulation Attack	Redundancy of GPS Modules, feedback, Consistency checks	Med	Med
External Spoofing of GPS Signal	Tippenhauer Method	Med-High	Low-med
Manipulated GPS Firmware Attack	Diverse Redundancy of GPS Modules and Verifiable Voting	Low-Med	Low-Med

Table 3. Impact of design patterns on the system.

The final steps in the process are to weigh the security trade-offs to determine which design pattern solutions are appropriate. The final steps are collaborative, all three teams return together and participate in a discussion regarding the security trade-offs that exist with the potential choices. While each defensive strategy remaining after step four has an acceptable impact on the attacker and on the defense, some may be better choices than others based on cost, effectiveness, and complexity. To carry out this assessment we go back to the influence diagram and instantiate the graph with the cyber-defense design patterns we have pre-selected. The annotated graphs shows where the design patterns are present in the system data-flow, what components are influenced by the additional hardware/software, and the security coverage of the design pattern with respect to the system as a whole. For sake of brevity, we provide a synopsis of the process in Table 4. The second row in the header indicates the specific attack scenario. The third header row indicates design patterns used to defense against the attack. The fourth header row indicates impact to designer (in terms of cost, and collateral impacts) and impacts to adversary. In the case of adversary impacts, the behavioral indication reflects the increase, decrease, or unchanged skills needed to carry out the attack after the design pattern has been added. This is an indication of the asymmetry effectiveness against threat actor. Overall, both design patterns increase asymmetry of the system. The trade-offs occur by examining what the designer's costs are for implementing the design patterns versus the cost to adversary to carry out the attack in view of the added defenses. In both cases, the costs to the adversary are significantly increased, while the system designer's costs are only modestly increased. In summary, both proposed design patterns are acceptable choices to carry forward.

Gimbal GPS Metadata Attack			
Embedded GPS Receiver Attack (Supply Chain)		**Down-stream GPS Malware Manipulation Attack**	
Diverse Redundancy and Verifiable Voting		**Redundancy of GPS Modules, feedback, Consistency checks**	
Impact to Defense	**Impact to Adversary**	**Impact to Defense**	**Impact to Adversary**
Implementation Cost: Low-Med	Design Knowledge: Increased to High	Implementation Cost: Med	Design Knowledge: Increased – Med-high
Collateral System Impact: Low-Med	Attack-Specific Technical Ability: Increased Med-High	Collateral System Impact: Med	Attack-Specific Technical Ability: Increased to Med-High
	Resources: Increased to High		Resources: Unchanged -Med
	Insider Presence (Operational): Increased to Med		Insider Presence (Operational): Increased - Low-med
	Insider Presence (Supply Chain): Unchanged High		Insider Presence (Supply Chain): Unchanged low
	Manpower/Time: Increased to Med-High		Manpower/Time: Increased- Med-high

Table 4. Effects of System-Aware defense on the system and attacker for the gimbal GPS metadata attack.

3.2.2.2 Introduction to GPS System Attack

Many autonomous and unmanned systems rely on GPS for navigation and control. This makes GPS an especially enticing target for the cyber-attacker. This attack scenario assumes that malicious hardware or software has been inserted into the GPS processor at some point along the supply chain. A triggering mechanism is included in this malicious hardware or software so that the GPS processor will report incorrect position information when triggered. The malicious deviation to the reported position will be introduced in a manner so that it is difficult to distinguish the malicious deviations from natural changes in position information.

The triggering mechanism may be external or internal to the GPS processor. An external trigger might arrive through an external radio channel or through conditions presented by other systems or sensors in the vehicle. An internal signal might be based on the sensed position. Thus, when the vehicle approaches coordinates stored in the GPS processor the malicious response is triggered.

While the malicious response could involve rapid and significant corruption of the position reported by the GPS, such rapid corruption of navigation data would be quickly and easily detected. A rapid and significant navigation disruption might be misinterpreted as a faulty GPS processor rather than a malicious attack, but either interpretation would prompt an immediate response to address the problem. Thus, the attack scenario anticipates that the malicious deviations will be introduced in a manner to make them difficult to distinguish from natural changes. An example of such masked deviations might involve a gradual introduction of error into the position data over a period of time so that the vehicle navigation system is slowly walked off of the correct position. The navigation system response would be to compensate for this slowly introduced position error in order to keep the vehicle on its intended course. These compensating corrections would slowly move the vehicle further and further off of its intended course.

3.2.2.3 High Level Description of GPS Attack

Two illustrative examples for this attack scenario have been prepared. The first of these examples is applied using HiL emulation in a laboratory environment. The second of these examples is applied to an autonomous vehicle in operation. This report describes the approach taken and the results obtained for the first of the two examples.

The first example uses the Piccolo autopilot hardware and software in the HiL emulation. The autopilot includes both GPS and INS components, but these systems are not used for the HiL emulation. Rather, this data is supplied by a simulator. Similarly, the autopilot normally provides control signals to actuators that control the behavior of the vehicle. During the HiL emulation, these control signals are sent to the simulation. Thus, the simulator establishes the initial position and orientation of the vehicle. The autopilot has a desired path to follow, and it sends control information to the simulator in an attempt to move the vehicle along this desired path. The simulator interprets the control information in the context of the vehicle capabilities and determines the updated position and orientation data that is sent to the autopilot. The particular autopilot used in this example communicates with the simulator using a CAN bus interface as described as described in section ⍰.

The second example will not be allowed to influence the trajectory of the vehicle directly because of safety concerns. However, there are enough other uses of position data on autonomous vehicles so that the scenario can be adapted to retain its attack capabilities while not compromising safety. The anticipated variation will apply the malicious and stealthily corrupted position to the metadata associated with captured images. Any live images linked from the vehicle to the base station will not be changed, but the changes in metadata will make it difficult to correlate captured data with cartographic databases. This attack and associated System-Aware protections were successfully demonstrated in the flight evaluations in October.

3.2.2.4 GPS System Attack Specifics

The first example uses HiL emulation in a laboratory environment. The particular autopilot used in this example communicates with the simulator using a CAN bus as illustrated in Figure 30.

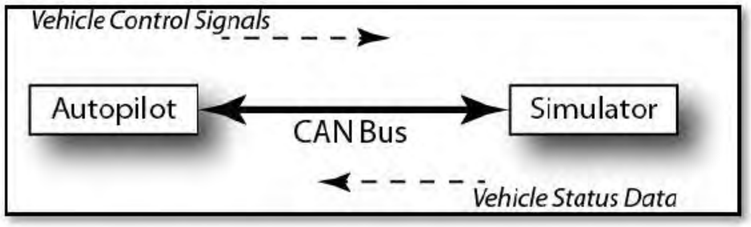


Figure 30. Autopilot to Simulator communications.

The figure shows that the control signals from the autopilot are conveyed to the simulator over the CAN bus. The simulator sends status data to the autopilot that would normally come from various sensors in the vehicle or in the autopilot. The GPS position data that would normally come from a GPS unit in the autopilot instead comes from the simulator through the CAN bus.

The CAN bus conveys data between devices in message frames. A frame contains header information along with the data in channels. For example, the message frame containing the GPS position data includes two channels: one for latitude and one for longitude. Each frame is distinguishable by its header information. Channels for each frame are then found in locations fixed for each message type within the data portion of the frame.

3.2.2.5 The Attack Simulation

The HiL emulation configuration suggests a direct path for implementing the example attack scenario. The simulation can be attacked by breaking the CAN bus between the autopilot and the simulator and inserting another device that can interpret and modify the message frames. This simulated attack configuration is illustrated in Figure 31.

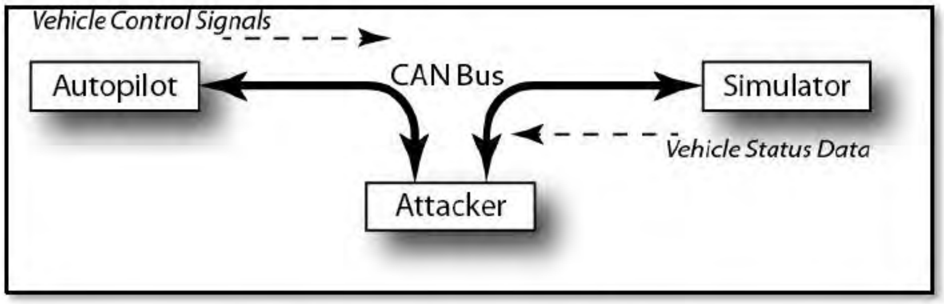


Figure 31. Simulated attack configuration.

The figure shows that all message traffic between the autopilot and the simulator passes through the attacker. For normal operations, the attacker forwards to the simulator all message frames sent by the autopilot. The attacker also forwards to the autopilot all message frames sent by the simulator. In this mode of operation, the HiL emulation proceeds as it would without the attacker in place. Initial experimentation confirmed that the HiL emulation proceeded as normal when the attacker simply forwarded all message frames in this manner.

The simulated attack requires that all message frames from the autopilot continue to be forwarded unchanged to the simulator. Also, message frames from the simulator must continue to be forwarded unchanged to the autopilot unless they are GPS position message frames. The attack requires that the attacker modify the data in the channels of the GPS position message frames before it is forwarded to the autopilot. These channels convey the latitude and longitude data to be corrupted.

The attack simulation was implemented using LabView running on a personal computer with two USB to CAN bus converters. This personal computer acted as the attacker. The physical CAN bus between the autopilot and the simulator was disconnected. The simulator CAN bus was connected to the attacker computer through one of the CAN bus converters. The autopilot was connected to the attacker computer through the other CAN bus converter. A LabView VI[1] was written to accept CAN message frames from both CAN interfaces and forward the frames received on each interface to the other interface. In this configuration, the simulation acted as normal.

The LabView VI in the attacker computer was then modified so that all CAN message frames continued to be forwarded as initially configured unless the CAN frames from the simulator were detected to be GPS position frames. The channels in these GPS position frames were decoded into latitude and longitude values and were written to a file. In addition, the LabView VI accepted latitude and longitude corruption value inputs that were added to the latitude and longitude values before the VI reassembled the GPS position message frame and sent it to the autopilot. Thus, the VI supported arbitrary adjustment to the GPS position values reported to the autopilot.

1 LabView is a graphical language that is proprietary to National Instruments. A LabView VI is a Virtual Instrument that is roughly equivalent to a computer programming routine.

As the attacker adjusted the reported GPS position, the autopilot adjusted the vehicle controls to correct the perceived position error. Thus, the autopilot drove the vehicle off of its intended course to correct the error introduced by the attacker. This is illustrated in Figure 32 which shows the deviation in the UAV's ground track.

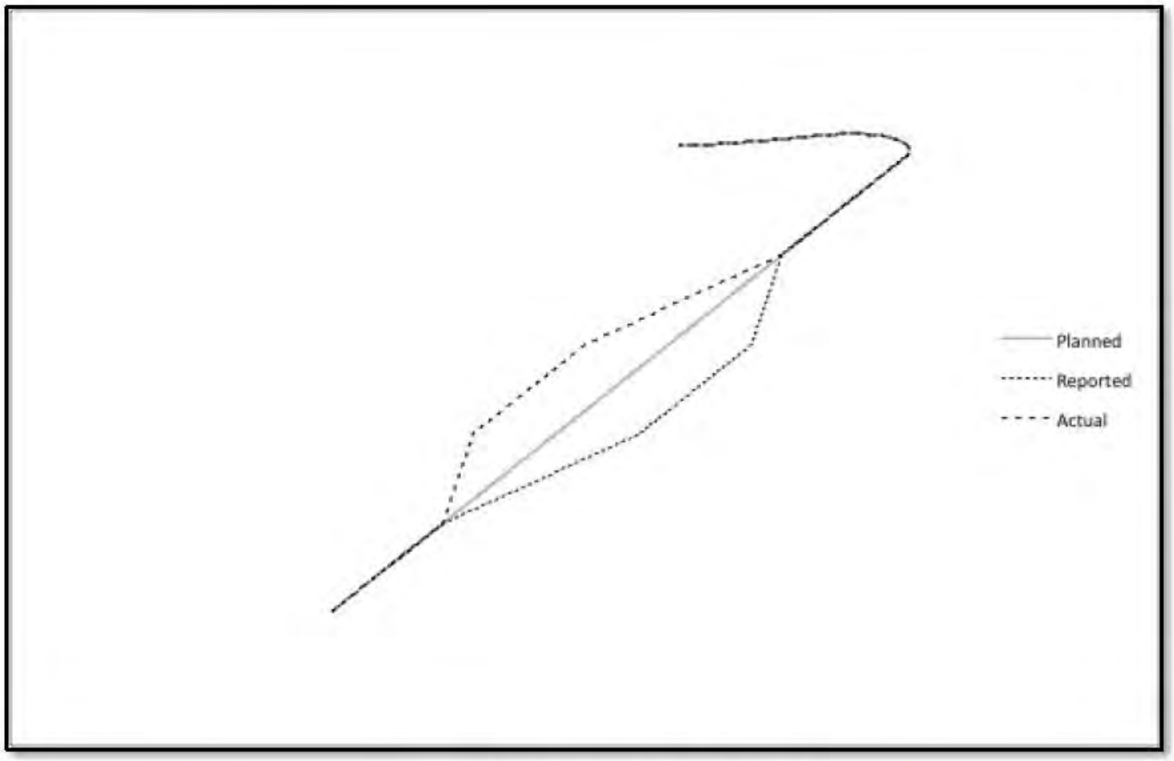


Figure 32. Example of GPS attack.

The figure shows the intended path for the vehicle using the solid gray line. At some point, the attacker starts moving the reported position down as shown in the figure so that the reported path deviates below the planned path. This is shown as a dotted line in the figure. The autopilot responds by adjusting the controls to keep the vehicle on the planned path. This actually causes the vehicle to deviate from the planned path in a direction opposite to the malicious change. This actual path is shown as a dashed line above the planned path in the figure. After the vehicle has moved beyond the area that the attacker wanted to protect, the attacker reduces the malicious deviation in the reported path until the vehicle returns to the planned pattern

3.2.2.6 Flight Demonstration - GPS Walk-Off Attack

For the flight demonstration, when the aircraft enters a pre-defined geographic region around the Early County Airport in Blakely, GA, an attack is triggered and the GPS location data from the autopilot to the camera gimbal is corrupted with a slowly increasing bias. The corrupted GPS data is included in the video stream as metadata that provides the coordinates of the area being viewed by the camera. As a result of this attack, the video imagery has erroneous position data associated with it.

The aircraft's commanded flight path was a right hand pattern on Runway 23 as shown in Figure 33. The geographic region for the attack was defined as a circular area centered to the Northeast of the aircraft's flight path with a radius of 609 m. This resulted in a partial overlap with the aircraft's commanded path as shown in Figure 33. As the aircraft crossed into the attack region the GPS walk-off attack was initiated. During the first pass through the attack region the Sentinel defense was disabled and the position data being fed from the autopilot to the gimbal was observed to deviate from the true position. During the next pass through the attack zone the defense was enabled and the GPS walk-off was detected almost immediately. The SerialSpy as described in Section 4.1, then replaced the corrupted GPS position data with validated data from the Sentinel and forwarded the data to the camera gimbal and into the video processing unit.

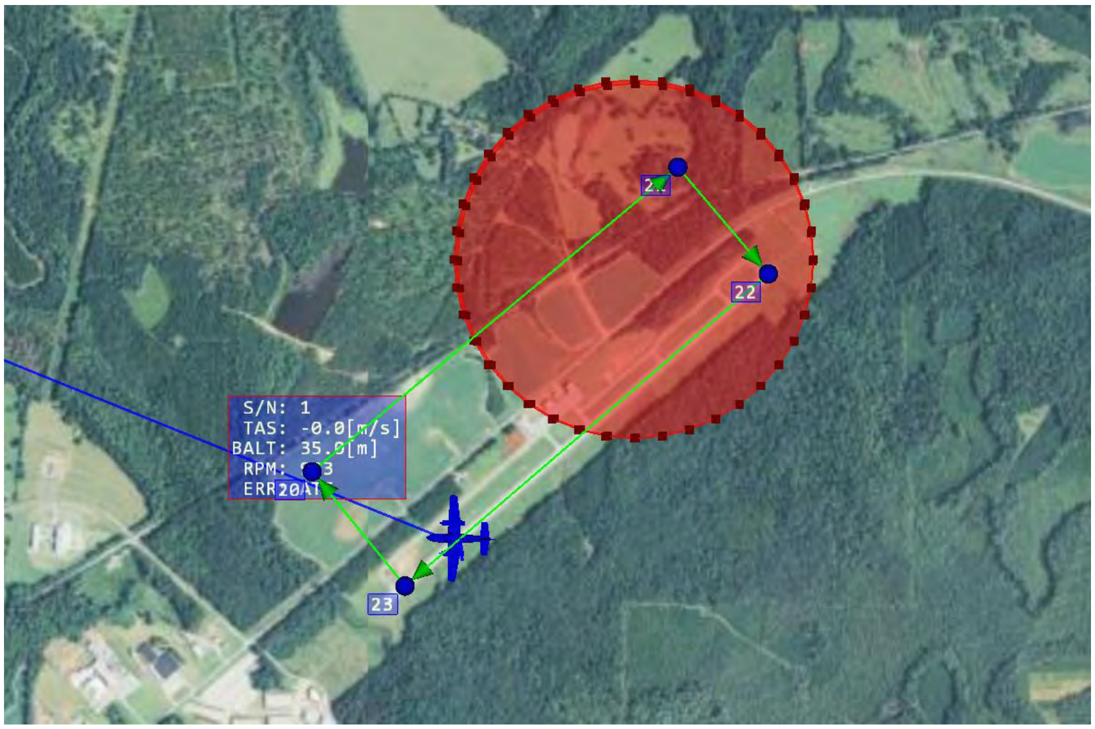


Figure 33. GPS walk-off attack region shown as upper right orbit. Typical flight path shown as right hand pattern on Runway 23

3.2.3 Gimbal System Control Attacks

3.2.3.1 Introduction to Gimbal Attacks

UAVs are predominantly used as ISR platforms carrying sensor payloads such as EO/IR cameras, synthetic aperture radar, signals intelligence systems, and others. As a result, sensor technology is evolving quickly, with new sensor systems being developed for all classes of UAVs. However, in the push to quickly field these new sensor suites and take advantage of their capabilities, cyber security is sometimes neglected. This creates an opportunity for an attacker to compromise a mission by exploiting weaknesses in the payload security; e.g., an attacker could degrade or deny the payload service or spoof the information coming from it.

To investigate methods for preventing, detecting, and countering potential cyber-attacks against UAV sensor payloads, the GTRI studied potential cyber-attacks and corresponding cyber security solutions for the TASE 150 camera gimbal system on its GAUSS UAV. The TASE 150 is a member of the popular and widely used family of TASE camera gimbal systems developed by Cloud Cap Technologies ™. The following sections describe potential attack vectors for the camera gimbal. Section 3.3.2.3 describes one approach to protect against these attacks.

3.2.3.2 High Level Description of Gimbal Attack

In order to determine the simplest vector to compromise the TASE camera gimbal, the GTRI analyzed the specifics of the TASE gimbal, the ViewPoint ground station software (used to view the video), and the communications protocol used to issue commands to the gimbal as well as receive status updates from the gimbal. This analysis revealed that the simplest attack vector would be to cause a denial of service or degradation of service by sending malicious, unauthorized commands to the gimbal from a malware exploit running on the operator interface machine (i.e., the machine hosting the PCC and ViewPoint).

This type of attack is possible because it is assumed that the source for all gimbal commands can be trusted. This means that as long as an attacker can communicate with the gimbal, she can have it execute any command that she wants. In addition, there are multiple commands that can potentially be exploited by an attack to cause a denial or degradation of service. Together, these factors suggest this path of attack.

The attack vector chosen for this study embeds a malicious exploit into ViewPoint. Embedding the malicious exploit is made possible by the open architecture of the ViewPoint and PCC software that allows developers to create plug-in software modules for added functionality. In addition, the PCC and ViewPoint allow users to go online and download maps and aerial imagery from several different map databases. No particular security measures are in place for users downloading maps onto the machine hosting the PCC or ViewPoint. Together these features provide a potential attack vector.

An alternative attack vector was considered that required communicating with the gimbal directly from a rogue wireless command tower. However, it was determined that the simplest solution would be to use the already established communication channel. In addition, solutions designed to detect malicious data sent from the operator interface should also be able to detect malicious data sent from an alternate source.

3.2.3.3 Gimbal Attack Specifics

The attack is an exploit embedded into ViewPoint that sends malicious data to the gimbal. The data will be unauthorized but properly constructed command packets designed to cause a denial or degradation of service. The exploit has the ability to construct the command data, compute the checksum, and send it to the gimbal. In addition to sending malicious data, the exploit can also produce non-malicious data at random intervals to attempt to hide the malicious data. The following are commands that could be used for a degraded or denial of service attack. For the flight demonstration, the Extend/Retract Gimbal command and the Gimbal command were chosen as examples of Denial-of-Service attacks that could be monitored by the Sentinel.

0x00 / 0x43: Extend/Retract Gimbal

By issuing commands to retract the gimbal during critical points in the mission, an attacker can cause the loss of a significant amount of information. By continuously issuing the command to retract the gimbal an attacker can cause a complete denial of service of the payload.

0x00 / 0x70: Disable Motor Driver

As with the Retract Gimbal packet, this command can cause a similar denial of service by interfering with the operator’s ability to steer the camera gimbal.

0x00 / 0x80: Gimbal Command

This command controls the location in which the gimbal is pointed. Pointing the gimbal away from the target can cause a denial of service. Random or erratic movement of the gimbal may cause the camera operator to assume a technical malfunction has occurred and recall the UAV.

0x00 / 0x40: Gyroscope Zero

This command sets the zero of the gyroscope on board the TASE gimbal. The gimbal documentation warns that the operator should not issue this command while the gimbal is in motion. Doing so may cause the gyroscope to be calibrated improperly, causing a degradation of service that would be difficult to fix mid-flight. This may force a recall of the UAV. The full extent to which this would affect performance has not yet been determined.

0x28 / 0x00: User Warning Packet

This packet is sent to the ViewPoint software instead of the gimbal. The software will display an error or warning message to the operator, which may be used to social engineer the operator into aborting the mission or taking other actions based on false information.

3.2.3.4 Flight Demonstration - Gimbal Command Attack

During the flight tests, when the aircraft entered a predefined geographic region, an attack is triggered and non-operator commanded gimbal commands were issued to the gimbal causing it to retract or slew the sensor point of interest (SPOI) upwards. The retract attack is referred to as Attack 2a and the SPOI attack is referred to as Attack 2b. The attacks appear to the operator as a malfunction of the gimbal mechanism and prevent him from conducting surveillance within the attack region.

The aircraft’s commanded flight orbit was centered over the runway as shown in Figure 34. The geographic region for the attack was defined as a circular area centered to the Northeast with a radius of 609 m. This resulted in a partial overlap with the aircraft’s commanded orbit as shown in Figure 34. As the aircraft crossed into the region encompassed by the attack orbit the SPOI attack was initiated

(Attack 2b). During the initial pass through the attack region the defense was disabled and the SPOI slewed upwards at the test director's command. During the next pass, the defense was enabled. The defense blocked the SPOI command preventing the camera from being slewed upwards.

After the SPOI attack was demonstrated it was deactivated and the gimbal retract attack (Attack 2a) was enabled. During the initial pass through the attack region the defense was disabled and the gimbal retracted when the operator tried to track an object at the center of the orbit. During the next pass, the defense was enabled. The defense blocked the gimbal retract command when the operator initiated a tracking command and the operator was able to successfully track a ground object.

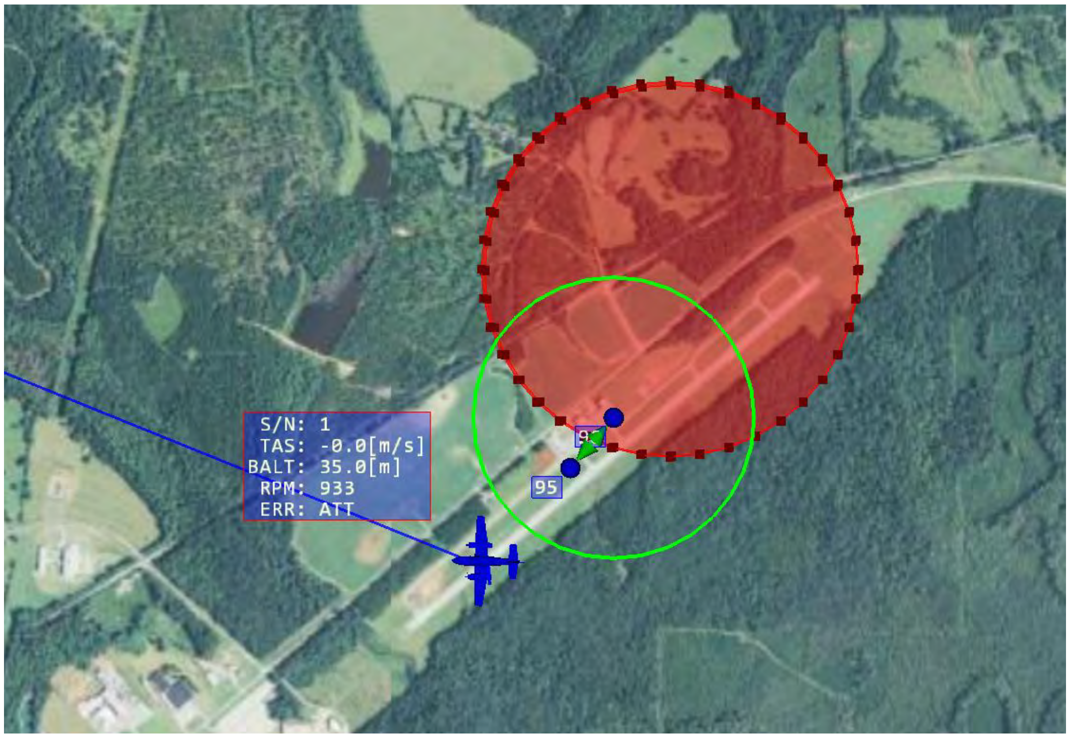


Figure 34. Gimbal command attack region shown as upper right orbit. Typical flight orbit shown as lower left orbit.

3.2.4 Hardware Security Against Design and Manufacturing Attacks

3.2.4.1 Introduction to Design and Manufacturing Attacks

Many attacks, including those outlined in section 3.1, could be injected into a UAV via the supply chain or by an insider that could embed malicious hardware:

- Designer adds malicious hardware functionality, which may not be detected in code review, IC inspection, and etc.
- Malicious functionality may be added in the fabrication process and escape detection in IC inspection.
- An attacker can reverse engineer unencrypted bitstreams to reveal the original design, or even modify the bitstream and add malicious functions.

- If a bitstream is encrypted using encryption algorithms such as AES, an attacker may still be able to decrypt it by using power analysis, or physically hacking into the device and obtaining the private key, which is used for encryption.
- An inside attacker may replace or modify the hardware that is ready for deployment.
- Attacks maliciously modify data during conversion between protocols; e.g., converting RS-232 to Ethernet.

For this project, we will focus on attacks against data protocol converters as potential attacks that maliciously modify data during protocol conversion not only serve as a vector for compromising an UAV, but also have the potential to compromise the Sentinel.

3.2.4.2 Flight Demonstration - High Level Attack Scenarios

As outlined in section 3.3 the prototype Sentinel converts information from RS-232 into Internet protocol (IP) packets. If an adversary could compromise this functionality, they could disable the protections afforded by the Sentinel. For example, assume that a Sentinel is monitoring a UAV's autopilot system. Furthermore, assume that all of the data is sent to the Sentinel using the RS-232 protocol and is converted by the Sentinel to Ethernet to simplify the implementation of protection algorithms. Now let us assume that an adversary has embedded a Trojan horse into the hardware performing the conversion on the Sentinel that looks for a specific pattern in the RS-232 data stream to trigger a denial of service attack against the Sentinel. As discussed in section 3.3, the protocol conversion will be implemented by running bare-metal applications on soft-cores implemented in a field-programmable gate array (FPGA).

A FPGA was selected for its configurability and flexibility that makes it favorable for the purpose of prototyping and concept proving. In this project, the soft-cores for protocol conversion and all hardware-based protections are implemented on FPGAs.

3.3 Design and Development of the Super Secure, Sentinel

This section outlines the implementation of a prototype super secure smart Sentinel to protect a UAV against the attacks outlined in section 3.1. The prototype Sentinel is capable of monitoring the autopilots subsystems, detecting when those subsystems have been compromised (i.e., when they have been altered through malicious activity), alert the appropriate authorities, and taking appropriate actions to restore those subsystems to an uncompromised state. For this project, whenever the Sentinel detects malicious activity it will alert a specially designated cyber officer responsible for ensuring the integrity of the UAV. To ensure that such an action cannot be intercepted by an adversary, the UAV has been equipped with a highly secured back channel that can be used by the Sentinel to communicate critical security information and ensure that the cyber officer is able to both receive information regarding the true state of the UAV, as well as continue to issue commands in the event the main communications channel is compromised.

3.3.1 Sentinel Platform Development

3.3.1.1 Single Board Computers

The Raspberry Pi, shown in Figure 35, is a 3.4” X 2.2” X 0.8” Single Board Computer (SBC) which has gained popularity because of its affordability (approximately $35). The design team chose the Raspberry

Pi as the major development platform for the flight test. This platform provides a relatively small, lightweight, and inexpensive option to use numerous platforms for snooping or corrupting serial data. The Raspberry Pi has a 700 MHz Armv7 based processor, and an SD card slot for memory storage. For our purposes, an 8 GB SD card is sufficient to host the Raspbian operating system, a version of Debian Linux specifically designed for the Raspberry Pi. The Raspberry Pi hosts C code written by the GTRI and the UVa teams which is responsible for byte-wise decoding of information as it is passed to and from the gimbal. It is also possible for the Raspberry Pi to host the communications software development kit (SDK) provided by Cloud Cap. This allows for easily maintainable and more legible C code to be used to decode the same information.

In addition to hosting the standardization of the data streams that are used by the Sentinel, the SBC's are also the platform where the code base for the System-Aware monitoring of the data from the system function(s) being protected, the detections based on the analysis of that data, the alerting mechanism to indicate the potential threat, and the corrective actions resides. Those algorithms are implemented in a combination of C code and Python code and are relatively small – under 600 lines of code each. This enables the Sentinel to provide multiple, diverse implementations of the code base in order to aid in the protection the Sentinel functions. In addition, the Raspberry Pi also allows for diversity in the operating systems (OS) in which the Sentinel logic resides. During the flight evaluations, both Debian-based Raspbian and Arch LINUX were implemented onboard the Pi's on the aircraft. This enabled the implementation of the diverse, redundant hopping security design patterns into the Sentinel implementation for the flight test. This software diversity introduces new challenges for the adversary should they choose to attack the Sentinel platform.

During the research effort, the use of SBC's from diverse vendors was also investigated in the bench top lab environment. We evaluated the Minnowboard – an Intel processor-based solution running Windows 8.1 and the Beaglebone Black – another ARM-based processor running Arch Linux as hardware and software solutions used to add diversity to the Sentinel solutions. The results of these tests showed that the Sentinel functions themselves were fairly agnostic to the hardware they were running on due primarily to the relatively small size of the code base. In addition, the diverse implementations of hardware and software environments offered low-cost solutions to increasing the complexity of the attack surface to an adversary tipping the symmetry of the attack space in favor of the system owner.

For the flight tests, however, the team decided to introduce only diversity in the operating systems, not in the hardware so only Raspberry Pi's we used in the actual tests. A set of three Raspberry Pi boards were implemented using triple modular redundancy in the processes for the System-Aware logic, and secured voting and random hopping were all used as security design patterns to protect the Sentinel while in operation in the flight tests.

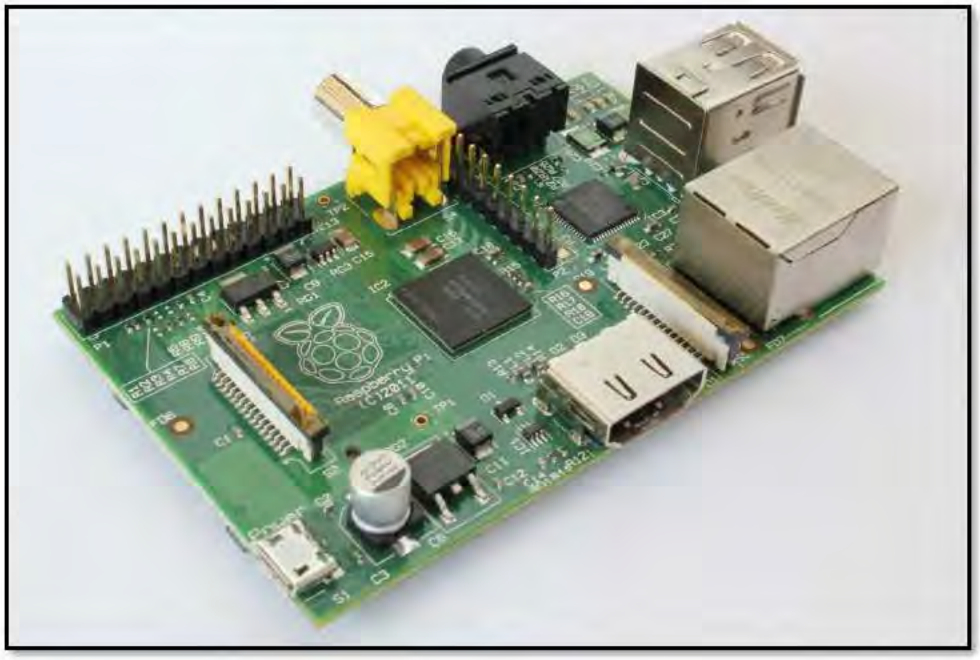


Figure 35. Raspberry Pi SBC.

3.3.1.2 SiCore SHIELD 2 Coprocessor

During Phase 1, the focus of the work performed with SiCore has been to design a solution that leverages the secure platform provided by the SHIELD card as a delivery mechanism for Sentinel functionality. The result of those activities is a new version of the SHIELD card called SHIELD II that serves as the central interface point between the system being protected, the UAV, and the Sentinel security design patterns that protect it. For the purposes of convenience for the demonstration of the Sentinel capabilities on this project, we choose to eliminate some of the hardened infrastructure for the SHIELD card and focused on adjusting the infrastructure of the card for two purposes:

1. Enabling the types of interfaces that are required to interface with the Piccolo autopilot system.
2. Protecting the data traversing the Sentinel architecture.

One of the goals in this effort was to look at delivering Sentinel functionality as a generic capability, while demonstrating that functionality on a specific system. To that end, we have decided to use IP as the standard protocol for Sentinel analysis functions. This particular system uses the serial RS-232 protocol for the majority of its inter-component communications. So, the conversion of RS-232 to TCP/IP becomes an important function and a potential area of vulnerability for attack. The design effort in this phase and described here reflect our desire to standardize the protocol and to protect that conversion process. This design should apply equally to other types of interfaces on other systems.

During Phase 2 the actual implementation and fabrication of the SHIELD 2 card was designed to meet the specifications described in section3.3.3.4 . That section will also detail the design decisions that were made during Phase 1. By using the SiCore SHIELD II card, we will be adding additional potential security features to the Sentinel including protections of the data bitstream, secured storage in the form of the a SD Card, securing the traffic within and outside the card and utilizing the OODA (Observe, Orient, Decide and Act) real-time controller methodology to aid in responding to events within the Sentinel security architecture.

3.3.2 Attack Detection, Mitigation and Restoration

3.3.2.1 Parameter-Based Attack Detection, Mitigation, and Restoration

To defend against the parameter-based attack outlined in section 3.2.1, a prototype super secure smart Sentinel that is capable of monitoring the autopilots parameters, detecting when the integrity of those parameters have been violated (i.e., when they have been altered through malicious activity), alerting the appropriate authorities, and taking appropriate actions to restore the integrity of those parameters (i.e., restoring them to an authorized state) has been integrated into the HiL emulation environment (see section 2.1). As seen in Figure 36, when the parameter integrity of the autopilot has been violated the Sentinel will alert a specially designated cyber security officer of the integrity violation. Several key factors affected the decision to send the information to a specially designated cyber security officer:

- Pilot Workload: We did not want to increase the pilot's workload further by making them responsible for deciding how best to respond to a cyber-security attack.
- Desired Response: There may be more than one way to respond to a cyber-security attack against the UAV, including allowing the attack to continue in order to gather information about the attacker's intention. A specially designated security officer would have the knowledge and experience necessary to work with the UAV flight crew to make those decisions.
- Insider Attack: It is possible that the attack was the result of an insider, possibly even the pilot of the aircraft themself! A special cyber security officer can make facilitate our ability to address the attack without alerting the insider.

These prototype efforts were taken forward into the design and implementation of the Sentinel that were flown in October.

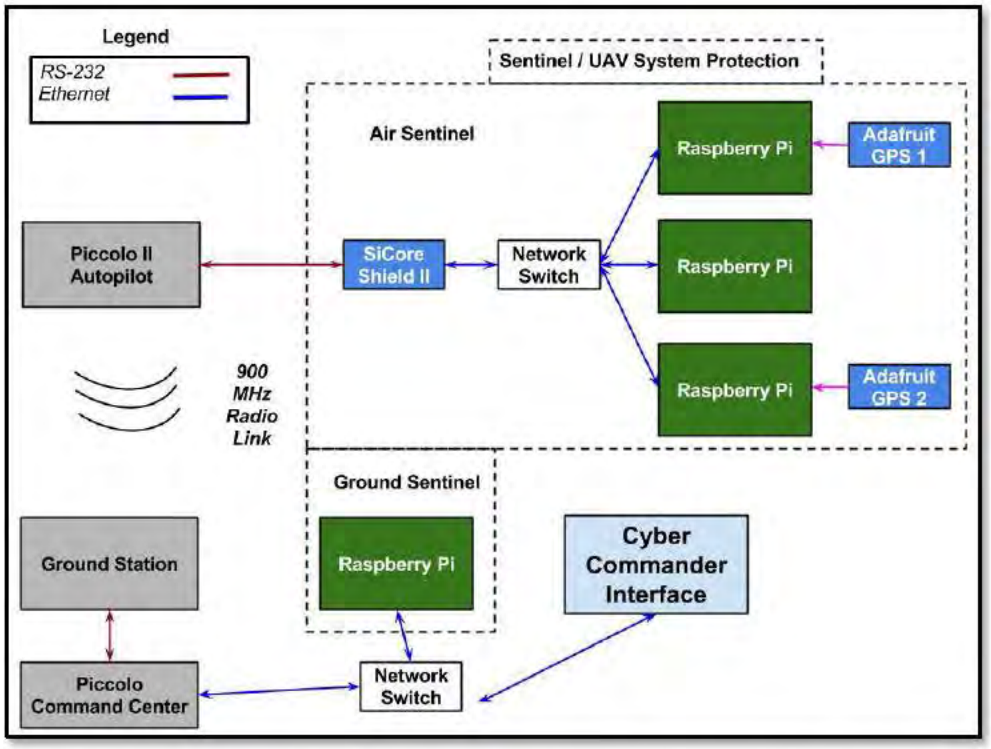


Figure 36. Super secure smart Sentinel for protection with a designated cyber security officer.

As the attack outlined in section 3.2.1 was focused on altering an UAV's flight plan by altering the UAV's waypoints, the initial prototype Sentinel will provide the cyber security officer (the Cyber Commander Interface) with information whenever a waypoint change occurs. In addition, as the attack could originate from the Piccolo autopilot or the operator interface, the cyber security officer will also be provided with information regarding where the malicious attack originated. For the initial prototype, the cyber security officer can respond to a cyber-attack in one of two ways:

1. Allow the attack to continue.
2. Restore the original flight plan.

In addition, to ensure that information sent by the Sentinel to the cyber security officer cannot be intercepted by an adversary, the UAV has been equipped with a highly secured back channel that can be used by the Sentinel to communicate critical security information and ensure that the operator is able to both receive information regarding the true state of the UAV as well as continue to issue commands in the event the main communications channel is compromised. For the initial prototype this security back channel is represented as a secure 802.11 network. This was replaced during the flight test to use the native flight radio available on the Piccolo II system.

The operator of the UAV may also make changes to the flight plan. As a result, the Sentinel must be capable of being able to distinguish changes authorized by the operator (i.e., legitimate) from changes made by the embedded Trojan horse (i.e., illegitimate). For the flight demonstration of the Sentinel, this was accomplished through the usage of an open source key logging program installed on the machine hosting the operator/pilot interface (i.e., PCC) in order to monitor the operator's inputs and send this information to the cyber security officer's work station. When the Sentinel protecting the autopilot detects a change in the flight plan, it will send an alert over the secure communications channel to the cyber security officer. This alert will then be cross-referenced against the inputs made by the operator for a corresponding change in flight plan. If an operator input for changing the waypoint is found, the cyber security officer is notified of an operator change in waypoints. If no operator input directing the UAV to another waypoint is found, then the cyber security officer is informed that a possible embedded attack has led to the UAV being directed to another waypoint. The cyber security officer is then presented with a list of options, which in the prototype only includes the option to restore the UAV to the original flight plan. If the cyber security officer decides to restore the aircrafts original flight plan, a message will be sent to the Sentinel over the secure communications channel and the Sentinel will restore the original flight plan, and alert the cyber security officer that the flight plan has been restored.

As outlined in section 3.2.1, a parameter-based attack may also be launched from a compromised operator interface (i.e., PCC). To protect against this attack an additional Sentinel was incorporated to monitor the data flowing into and out of the PCC.

3.3.2.2 GPS System Attack Detection and Mitigation

Before we can describe our Phase 2 detection and attack mitigation methodologies for the GPS system, we must first explain the proposed architecture of the diversely redundant navigation components in the Sentinel. Next, we describe the analytical tools used to improve the system's resiliency under an attack. We explain how these components are implemented and how these components work together. Third, we outline the recovery procedures built upon the analytical tools. Finally, we list the benefits of such an approach—including the speediness of recovery compared to traditional methods and capturing information on the adversarial strategy and motive.

We implemented an architecture of several stand-alone navigation systems (i.e., GPS INS, and the Piccolo navigation system) or the addition of GPS sensors from multiple vendors and hence, different manufacturing vendors and processes. We have attached these extra redundant navigation systems into the Sentinel as shown in Figure 5. Each of these components carries diverging algorithms for navigation. The de facto Piccolo II navigation system uses an INS and a GPS in tandem to give the autopilot the estimated location of the aircraft. The strap-on INS calculates the aircraft location based on accumulated data from motion sensors (accelerometers) and rotation sensors (gyroscopes) to estimate the aircraft's position. The GPS uses time signals from multiple GPS satellites to triangulate an aircraft's location.

To supplement the de facto navigation system of the Piccolo II, we decided to add secondary GPS units. These units are supplied by a vendor different than those embedded into the Piccolo II and connect directly to the Sentinel. We use these components to verify the behavior of each component to see if one or more of these components are performing anomalously and also to give the system a fallback source of location data. Note that if further flight tests are conducted, we would augment the use of GPS systems with another source of location data such as non-GPS assisted INS/IMU as a source for verification of the GPS streams and as a potential fallback location data source.

In order for an adversary to successfully exploit a UAV navigation system, they must be able to simultaneously manipulate all sensory information. Diverse redundant components—like the ones previously described—have the potential to increase the difficulty and cost (time, resources, and labor) to the adversary.

3.3.2.2.1 Frame of Discernment

The purpose of the Frame of Discernment (FOD) is to enumerate the exhaustive and mutually exclusive scenarios. For our purposes, Table 5 shows the FOD of our UAV navigation architecture. The columns represent each stand-alone component and the rows enumerate the possible 3^2 events. The red cell indicates an attack on its indicated component, while a yellow cell indicates a proper functioning component. For example, we define Event 1 as the event where all components are reliable and functioning as expected. Event 2, in contrast, is defined as the event where the Piccolo II is manipulated. The Frame of Discernment organizes the unobservable and inscrutable events into one in which we could compute and compare each individual event's likelihood of taking place given the observable live data streaming from these components.

EVENT	P	INS2	DGPS
1	0	0	0
2	1	0	0
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	1
7	0	1	1
8	1	1	1

Table 5. Frame of Discernment for Navigation Architecture.

3.3.2.2.2 Similarity Measurement

In this section, we tie in the concept of Similarity Measurement procedures with the Frame of Discernment. Similarity measurements quantify the compactness and intimacy of the streaming sensor readings against another sensor. Under this similarity procedure, we should be able to adapt to the variability of error the sensor measurement with one another (See Appendix: Proposed Mass Function for FOD).

Let us define Gaussian random variables $\mathrm{X_t}$, $\mathrm{Y_t}$, $\mathrm{Z_t}$ to represent the values of the Piccolo navigation system, INS, and GPS respectively at time t. We use these random variables as elements to measure the mass function for each the events in the FOD. As a candidate mass function, we choose for Event 1:

$$m_{1,t}(X_t = x, Y_t = y, Z_t = z) = \left[\frac{prob(x-y)}{\max prob(x-y)}\right] * \left[\frac{prob(x-z)}{\max prob(x-z)}\right] * \left[\frac{prob(z-y)}{\max prob(z-y)}\right]$$

The mass function will decrease as one of these random variables deviates from one another. The maximum value of $m_{1,t}$ is 1 (for the case $x = y = z$), and the minimum value is 0. We develop a list for candidate mass function for each event in the FOD. To examine the mass functions for the rest of the events in the FOD, refer to appendix 6.1 at the end of this report.

3.3.2.2.3 Analytical Equivalent Pairings

There exists analytical pairings in the FOD in which events are indiscernible with each other. These events share an identical mass function. For example, Event 2 and Event 7 is an analytical pairing in which we cannot discern if the Piccolo II's navigation system is, or simultaneously both the INS and GPS, are attacked. Although we cannot distinguish which of the event is occurring, we can palliate such occurrences by adding additional N redundant navigation components to the system; thus extending the FOD from 2^3 to 2^{3+N} events. In effect, the mass functions have to adapt to the additional N components.

For example, if $N = 2$, adding an barometric altimeter (ALT) and a location estimator (EST), then the FOD table becomes:

EVENT	P	INS2	DGPS	ALT	EST
1	0	0	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0
6	1	1	0	0	0
7	1	0	1	0	0
8	1	0	0	1	0
9	0	1	1	0	0
10	0	1	0	1	0
11	0	0	1	1	0
12	1	1	0	1	0
13	1	0	1	1	0
14	0	1	1	1	0
15	1	1	1	0	0
16	1	1	1	1	0
17	0	0	0	0	1
18	1	0	0	0	1
19	0	1	0	0	1
20	0	0	1	0	1
21	0	0	0	1	1
22	1	1	0	0	1
23	1	0	1	0	1
24	1	0	0	1	1
25	0	1	1	0	1
26	0	1	0	1	1
27	0	0	1	1	1
28	1	1	0	1	1
29	1	0	1	1	1
30	0	1	1	1	1
31	1	1	1	0	1
32	1	1	1	1	1

Table 6. FOD with Altimeter and Location Estimator.

Using 5 navigation components, to completely control an aircraft, the adversary needs to manipulate 3 components simultaneously—increasing the difficulty of success. By increasing the number of components to 5 and augmenting the elements of the FOD to $2^5 = 32$ events, we increase the difficulty of success for the adversary by forcing the adversary to capture 3 components. Even if the adversary successfully infiltrates the majority of the components, UAV managers have enough evidence to flag it as a major attack and shut down the flight mission.

3.3.2.2.4 Sequential Change Detection

We propose a sliding window of size W to estimate the mass function of a system. A simple average procedure would be a reasonable and effective method for estimating the mass functions. However, it is possible to use a Likelihood Ratio (GLR) algorithm to estimate the mass functions provided that we have a variance matrix Q for each mass function. We can find Q using empirical tests in normal flight i.e., flight without attacks.

The estimate then is

$$\widehat{m}_j(t) = \frac{\mathbf{1}Q^{-1}(M_j^t - \mu_0)}{\mathbf{1}Q^{-1}\mathbf{1}}$$

Where for each j in the FOD and 1 is the vector of ones with size equal to that of M_j^t and

$$M_j^t = \frac{1}{t+1}\sum_{i=t-W}^{t} m_{j,i}$$

The event with the greatest $\widehat{m}_j(t)$ is the event most likely occurring. Choosing a larger W would slow down the identification of $m_{j,t}$ and a narrow W would result in a noisier $m_{j,t}$ for each element j in the FOD.

As our recovery protocol, we create predetermined procedures for each element in the FOD

Suppose we have updated estimated mass functions with a sliding window of size W for each time t. The event in the FOD with the highest mass function signifies the event with the highest likelihood of taking place. For each event, we map a predetermined navigation procedure.

EVENT	P	INS2	DGPS		VERSION	PROCEDURE	
1	0	0	0		1	y(t) =	P
2	1	0	0		2	y(t) =	INS2,DGPS
3	0	1	0		3	y(t) =	P
4	0	0	1		4	y(t) =	P
5	1	1	0		5	y(t) =	Failure
6	1	0	1		6	y(t) =	Failure
7	0	1	1		7	y(t) =	Failure
8	1	1	1		8	y(t) =	Failure

Table 7. Navigation procedures.

For 5 components, the proposed procedure becomes

EVENT	P	INS2	DGPS	ALT	EST	VERSION	PROCEDURE	
1	0	0	0	0	0	1	y(t) =	P
2	1	0	0	0	0	2	y(t) =	INS2,DGPS
3	0	1	0	0	0	3	y(t) =	P
4	0	0	1	0	0	4	y(t) =	P
5	0	0	0	1	0	5	y(t) =	P
6	1	1	0	0	0	6	y(t) =	DGPS
7	1	0	1	0	0	7	y(t) =	INS2
8	1	0	0	1	0	8	y(t) =	INS2,DGPS
9	0	1	1	0	0	9	y(t) =	P
10	0	1	0	1	0	10	y(t) =	P
11	0	0	1	1	0	11	y(t) =	P
12	1	1	0	1	0	12	y(t) =	Failure
13	1	0	1	1	0	13	y(t) =	Failure
14	0	1	1	1	0	14	y(t) =	Failure
15	1	1	1	0	0	15	y(t) =	Failure
16	1	1	1	1	0	16	y(t) =	Failure
17	0	0	0	0	1	17	y(t) =	P
18	1	0	0	0	1	18	y(t) =	INS2,DGPS
19	0	1	0	0	1	19	y(t) =	P
20	0	0	1	0	1	20	y(t) =	P
21	0	0	0	1	1	21	y(t) =	P
22	1	1	0	0	1	22	y(t) =	Failure
23	1	0	1	0	1	23	y(t) =	Failure
24	1	0	0	1	1	24	y(t) =	Failure
25	0	1	1	0	1	25	y(t) =	Failure
26	0	1	0	1	1	26	y(t) =	Failure
27	0	0	1	1	1	27	y(t) =	Failure
28	1	1	0	1	1	28	y(t) =	Failure
29	1	0	1	1	1	29	y(t) =	Failure
30	0	1	1	1	1	30	y(t) =	Failure
31	1	1	1	0	1	31	y(t) =	Failure
32	1	1	1	1	1	32	y(t) =	Failure

Table 8. Procedures for five navigation components.

If we have 5 components, we force the attacker to be capable of manipulating 3 components for her to be successful.

This approach improves the resiliency and reliability of the Navigation System and increases the difficulty to attain success and provide the managers information on the adversarial strategy and motive.

Using this architecture with the proposed protocol, we are able to increase the difficulty of adversarial success. The adversary is required to successfully manipulate the majority of the components. The Frame of Discernment enables the user to compare events based on mass functions and detect which event has the maximum likelihood of occurring. The FOD also organizes which recovery procedure to choose in order to isolate the component under attack.

Also, using similarity measurements, we improve the recovery speed compared to simple threshold procedures—which gives adversaries room to manipulate the aircraft, and give false negative and false positive in noisy systems if the threshold values are incorrectly provided.

By allowing the adversary to freely manipulate the sensors without shutting down the flight mission, we can gather information relating to the adversarial attack strategy and adversarial motive—which may be valuable to managers and strategists. This is another important feature that the proposed procedure provides which threshold methods do not immediately and directly deliver.

For Phase 2, we will apply the methods described above to enhance the security of the UAV flight camera metadata and will utilize three sources of GPS data coordinates – the native GPS from the Piccolo II and the 2 new GPS sensors added to support the Sentinel.

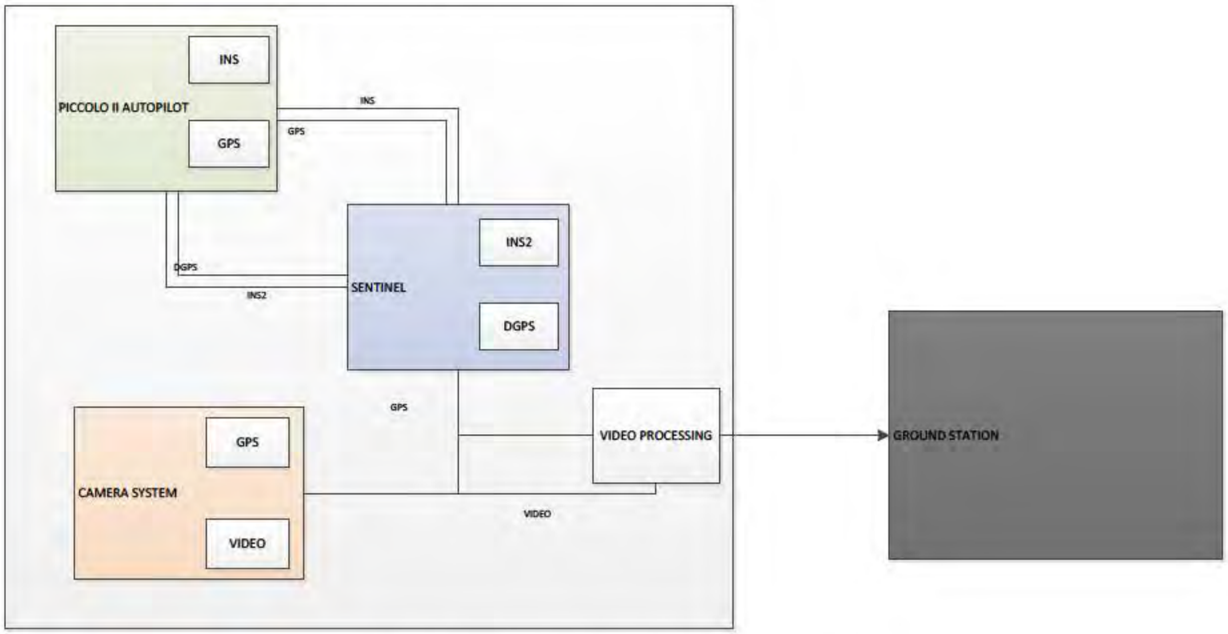


Figure 37. Architecture for camera system.

Figure 37 summarizes the proposed defense architecture for the flight camera system of the UAV. The box encapsulating the Piccolo II, Sentinel, and camera system modules carried on the UAV. The camera outputs two types of streaming signals: video and metadata associated with the video stream. The metadata includes GPS coordinates of the streaming files, which an adversary could manipulate via corrupting CAM-GPS system in the camera.

We propose to use the defense procedures for the navigation system for the use of securing the metadata. We will continue to use diverse, redundant navigation components—diverse GPS systems housed in the Sentinel, and the GPS housed in the camera system. Also available for use is the GPS and INS navigation system for the Piccolo II.

Table 9 below enumerates the events in the FOD.

EVENT	P	INS2	DGPS	CAM-GPS
1	0	0	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	1	0	0
7	1	0	1	0
8	1	0	0	1
9	0	1	1	0
10	0	1	0	1
11	0	0	1	1
12	1	1	0	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	0
16	1	1	1	1

Table 9. FOD for camera system.

We will continue to use the same type of mass functions for each of the events in the FOD. This time, however, we have four components. Let W, X, Y, Z be Gaussian random variables representing the navigation measurements for each of the sensors. Then the mass function for Event 1 where all components are reliable is

$$\begin{aligned} & m_{1,t}(W_t = w, X_t = x, Y_t = y, Z_t = z) \\ & = \left[\frac{\text{prob}(w-x)}{\max \text{prob}(w-x)}\right] * \left[\frac{\text{prob}(w-y)}{\max \text{prob}(w-y)}\right] * \left[\frac{\text{prob}(w-z)}{\max \text{prob}(w-z)}\right] * \left[\frac{\text{prob}(x-y)}{\max \text{prob}(x-y)}\right] \\ & * \left[\frac{\text{prob}(x-z)}{\max \text{prob}(x-z)}\right] * \left[\frac{\text{prob}(z-y)}{\max \text{prob}(z-y)}\right] \end{aligned}$$

Again, we use the recovery procedures outlined in Phase 1 to determine which signals the Sentinel should be allowed to send. Below is the recovery procedures linked with each event in the FOD.

EVENT	P	INS2	DGPS	CAM-GPS	VERSION	PROCEDURE	
1	0	0	0	0	1	y(t) =	CAM-GPS
2	1	0	0	0	2	y(t) =	CAM-GPS
3	0	1	0	0	3	y(t) =	CAM-GPS
4	0	0	1	0	4	y(t) =	CAM-GPS
5	0	0	0	1	5	y(t) =	DGPS
6	1	1	0	0	6	y(t) =	DGPS
7	1	0	1	0	7	y(t) =	CAM-GPS
8	1	0	0	1	8	y(t) =	DGPS
9	0	1	1	0	9	y(t) =	Failure
10	0	1	0	1	10	y(t) =	Failure
11	0	0	1	1	11	y(t) =	Failure
12	1	1	0	1	12	y(t) =	Failure
13	1	0	1	1	13	y(t) =	Failure
14	0	1	1	1	14	y(t) =	Failure
15	1	1	1	0	15	y(t) =	Failure
16	1	1	1	1	16	y(t) =	Failure

Table 10. Procedures for camera system.

3.3.2.2.5 Flight Demonstration - Navigation System Protection

During Phase 2 of the project, the security design patterns that were designed for the protections of the in Phase 1 were implemented to protect the navigation and imagery metadata collection systems which rely on GPS navigation data from attacks which would either walk-off the aircraft to an undesired location or corrupt the GPS location metadata on the imagery data that is being collected by the surveillance platform. For safety reasons, it was decided to use the imagery data corruption in the flight-test scenarios rather than risk losing control of the aircraft. However, the same attack methods and protections would be applied in either case.

The basic configuration for the GPS navigation system on board the GAUSS UAV simply consisted of one GPS device integrated in to the autopilot system on board the aircraft. This device was solely responsible for plotting the aircraft's location coordinates in the air and on the ground and also for providing GPS location information to the payload video collection system that tags the video data with location metadata. In the case of a cyber-attack emanating from a supply-chain or from an insider-based attack where perimeter defenses have been penetrated, there was no system in place for defending the aerial vehicle and protecting the GPS location data that is critical to systems throughout the surveillance platform. In Phase 1 we sought to increase the security of the navigation system by bringing in System-Aware Cybersecurity strategies and developed a security design pattern which implemented inexpensive, redundant location sensors to the UAV's systems and proposed a strategy for comparing location information from those diverse data sources and a secured voting procedure for eliminating potentially compromised sources of data. This method could include the use of diverse types of location information coming from systems other than GPS such as Inertial Navigation Systems (as proposed in

Section 3.3.2.2) thus addressing situations where the entire GPS system has been attacked whether through a jamming or a spoofing attack. For the flight evaluations, we determined that the specific attack we would address was focused on an attack on a particular chip set commonly used in GPS receivers. We added redundant, inexpensive ($100 each) GPS receivers from diverse manufacturers to the Sentinel in order to increase the complexity for an adversary who is attempting to corrupt the supply chain for GPS systems.

Two additional GPS sensors were added to the existing sensor suite on board the GAUSS to sit alongside the existing autopilot GPS. Each of these new GPS sensors was plugged into the secured environment of the Sentinel via a USB ports as shown in **Figure 36**. These are listed as Adafruit GPS1 and GPS2. For the flight evaluations, it was decided that the Sentinel configuration would be made up of three components that were Single Board Computers (SBC) running diverse operating environments. Furthermore, two of the SBC's were augmented with the diverse, redundant GPS receivers. This provides at least three sources of data that is a requirement for our data comparison and voting algorithms. Those SBC's ran a small data receiver program that collected GPS data in the form of "(TIMESTAMP, LATITUDE, LONGITUDE)". Each of the receiver programs then sent this data to other logic programs on each of the three SBC environments which are in turn responsible for taking the outputs from all three devices (Autopilot GPS, additional GPS receiver 1, and additional GPS receiver 2), and the latitude and longitude values from each of the three sensors and running the Sentinel data comparison algorithm to check for irregularities between the data streams. In addition, the algorithms are small, on the order of approximately 500 lines of code or less, and are implemented in multiple programming languages (C and Python). This diversity makes the Sentinel logic much more difficult to attack. Any location data streams that are showing irregularities within the boundaries of the tuning parameters of the algorithm are voted out on each SBC. If there is no discrepancy in the streams, the SBC randomly chooses the current source for the data. Furthermore the output of the logic program from each board is then sent to another voting algorithm and the values are again compared. Again, if all are in agreement, the secured voter randomly chooses from the available location sources for the system to use. The system can also be configured to allow the main system GPS to be used unless an attack has been detected. If there is a discrepancy detected, then the potentially corrupted stream is voted out and the new, reliable source of location data is randomly chosen from the remaining sources.

3.3.2.2.6 Issues We Addressed in the GPS Development Efforts

We ran into several issues with augmenting the Sentinel with the additional GPS sensors and integrating the information from the new sensors into the existing systems we were monitoring on the aircraft. First, there was a difference between the frequencies of the sampling rates of the GPS sensors we added and the existing GPS information flowing through the Piccolo autopilot. The two external GPS sensors that we acquired would send data to the receiver program once per second and the receiver program would then pass that data to the logic program at the same rate; whereas the GPS data coming from the autopilot system was sent to the logic program in "chunks" every 5 to 6 seconds. This complicated the comparison process. Instead of being able to constantly compare data between sensors, we had to make the comparisons an event-driven process so that we were only doing comparisons when we received a chunk of new location data from the autopilot. We designed code within the logic program that buffered the data from the two external sensors. When a chunk of 5 seconds of autopilot data came in, we would take the autopilot data from each second and compare its timestamp. Then we would search the buffers of both external GPS sensors, searching for a timestamp match. When we found a match, we would then compare the latitude and longitude values that corresponded to this timestamp from each sensor. This process would be done for all 5 seconds of data within the chunk sent by the autopilot.   This led us to a second issue related to the timestamp format of the different GPS receivers. This was relatively simple to fix, but the two external GPS devices used Unix time for their timestamp, whereas the existing autopilot GPS sensor used Coordinated Universal Time. Code was deployed in order to change the format of the autopilot GPS timestamp, so that Unix time became our system-standardized form for time.

Once our algorithms were implemented in order to collect the data and send it along using the receiver programs, we then needed to set the standard for what the logic programs would define as a sensor behaving “irregularly”. For example if the output of one sensor varied from the other two sensors by one-thousandth of one degree of latitude, would we consider that “irregular” or simply account that to GPS noise that is inherent with these sensors? The detection algorithm that we developed was entirely based off the Master’s thesis of J. Vincent Pulido (2014) entitled, *A Method for the Detection and Diagnosis of Stealthy False Data Injection Attacks in Cyber-Physical Systems* and in described in Section 3.3.2.2.

The first step in the process is “calibrating” the detection algorithm to our specific sensors. This involved finding the standard noise for each of the three GPS sensors. To do this we set the three sensors in a fixed location and collected and stored data from each for an hour at a time. We then simply calculated the residual values by simple subtraction. That is, we took the latitude value from additional GPS1 at each second, and subtracted the latitude value from additional GPS2 sensor at the corresponding second. The same process was done for longitude. The mean residual value was calculated, as were the variance and the standard deviation. The mean, variance, and standard deviation residual values are essential for calibrating the algorithm that described in his the Pulido thesis. Data from additional GPS2 sensor was compared with the autopilot GPS to create the mean, variance, and standard deviation residual values. These values were included into our detection method, and were used to standardize latitude and longitude residual values between the sensors. Once calibrating the algorithm is done, you are then ready to start utilizing the detection algorithm.

Essentially, the detection algorithm operated by evaluating a chunk of data as it passes from the autopilot GPS, and the data from all three sensors are compared, and the residuals are calculated. These residuals are then standardized using the mean and standard deviation values that determined during the calibration process. The detection algorithm then runs a hypothesis test using a Chi-Squared distribution. The alpha value chosen for this hypothesis test, by default, is .01. However the user is able to increase or decrease this value at any point in order to obtain a desired false alarm rate and sensitivity for a certain mission. A challenge that we ran into when implementing the algorithm was the complexity of actually conducting a hypothesis test with a Chi-Squared distribution. Do to the mathematical complexity, we found it convenient to simply put the entire GSL Scientific Library that contains a Chi-Squared function on the single-board computers that housed the logic programs. This library would need to be validated as secured in operational use.

The project team determined that in the event of an algorithmic detection, the system would only warn the cyber commander of a potential cyber security attack if one device "consistently" behaved irregularly. That is, if a device behaves irregularly just a few times, then we are willing to write the anomalous behavior off as GPS noise. In order to define a consistent behavior, we established a "sliding window" evaluation method within the program logic. The sliding window consists of two user-adjustable parameters: window size and condition. The value that is set for the window size tells the program how many seconds of data the algorithm will consider. The condition value controls when an alert will actually be sent to the ground. For example, if you set the window size to be 60 and the condition to be 50, then one minute of data will be viewed (60), and if 50 of those 60 readings say that a specific GPS sensor was behaving irregularly, then that will register as a detection and will indicate an attack. If the user does not set sliding window values, then the program defaults to having a window size of 30 and a condition value of 25. Choosing the ideal window size and condition values is something that could be researched further in the future as experiments with the algorithm are applied to different system types.

In the current system and by default the final latitude and longitude values that are sent to the ground are the data from the autopilot GPS sensor. If the autopilot data consistently behaves irregularly, the ground station will be notified of the potential attack and will be given the choice to do nothing and continue using the autopilot data, or to take restorative action and switch to the coordinates from one of the external GPS sensors. If the cyber commander elects to take restorative action, the system will randomly select between the two external sensors and output their latitude and longitude values instead.

In order to add an additional layer of diverse redundancy, we chose to not only have multiple sensors collecting data, but also to have three single-board computers running the comparisons independently of each other. Two of these boards were attached to the additional GPS sensors, and the other simply sat alongside them. The two boards that were connected to the additional sensors were each given the receiver program. All three of the boards were given the logic program. In addition to these two programs, the use of three single-board computers required us to build a voting program, and put that program on each of the three boards.

Essentially, the receiver programs would collect the timestamp, latitude, and longitude data from the two additional GPS devices. The receiver program then sends out the coordinates using the RabbitMQ messaging system (described in Section 4.7.7). Each of the three boards would then use their logic program to gather the data from all three GPS devices. The logic program then conducts the detection algorithm, and adjusts the sliding window. The logic programs output a string containing "(TIMESTAMP, LATITUDE, LONGITUDE, ATTACK_INT)". Attack_int is simply a way of displaying, which, if any, of the GPS devices were under attack. Attack_int is capable of holding the values 0, 1, 2 ,and 3. For example, if the attack_int equals 0, then no devices are under attack. However, if attack_int were to equal 1,then that would mean that additional GPS1was consistently behaving irregularly. Similarly, if it were to equal 2, additional GPS2 would be considered under attack; and if attack_int were to equal three then the autopilot GPS would be consistently behaving irregularly.

The outputs from all of the three single-boards are sent to a voting program, where it is determined what data from which computer will actually be sent to the ground. If the outputs from all three logic programs are exactly identical, then the voting program will randomly select the data from one of the three boards, and send that information out to the rest of the system. If the three outputs are not an identical match, then they will be put through an algorithm nearly identical to that of the sensor detection algorithm that is described above. If this detection algorithm determines that all of the data seems regular, then the voting program will again randomly select the data from one of the three boards to output. However if the algorithm determines that one of the boards is outputting bad data, then it will randomly select between the data of the other two boards, and send an alert to the ground warning them of a potential single-board computer attack. The voting program outputs a string containing "(TIMESTAMP, LATITUDE, LONGITUDE, ATTACK_INT, SINGLE_BOARD_ATTACK, DEVICE_USED)", this string is sent to the ground station. Single_board_attack is capable of equaling the values 0, 1, 2, and 3. If all the single-board computers' logic algorithms appear to be outputting similar or equivalent data, then single_board_attack will equal zero. However, if one single-board computer is consistently behaving irregularly, then the single_board_attack value will be 1, 2, or 3; depending on which computer is considered to be under attack. The device_used field is capable of being 1, 2, or 3. This value tells us which single-board computer's data is actually sent to the ground station.

In order to provide even more diversity, we created the voting program so that it hops between the single-board computers. The voting program is housed on all three single board computers, and one of them is randomly selected to run the voting program. A new voter is selected every five seconds. That is, no one board is responsible for conducting the voting.

One additional security measure that we took was to code the boards in different programming languages. So in our current setup, two of the boards solely have code written in C. One of the boards, however, has their logic and voting programs written in Python. The issues that we ran into include altering code structure, and finding a suitable replacement for the GSL Scientific Library that houses the method required for conduction the Chi-Squared hypothesis test.

There are several areas where additional research can be conducted. First, the entire system relies heavily on the RabbitMQ messaging system. Additional protections and securities for this system should be researched. Second, instead of solely comparing GPS data, it would be more secure to also acquire additional, external accelerometers. These accelerometers could be used to validate the GPS data to add an additional layer of security. Third, while collecting an hour's worth of data was suitable for our tests of calibration purposes, it may be desirable to collect more data to refine this system. It should be researched, how many data points are actually required to have a fine-tuned detection algorithm. Lastly, it may also be desirable to research a self-adjusting alpha value for our detection algorithm. In poor weather the current algorithm often produces many false alarms. It could be advantageous if the alpha value in our Chi-Squared hypothesis test automatically adjusted depending on the number of satellites that each GPS sensor had a “fix” on. Research regarding what alpha value should be chosen given certain weather circumstances should be looked into.

3.3.2.3 Gimbal Attack Detection and Mitigation

Degradation and denial of service attacks are possible because the gimbal trusts the sender of any commands that it receives. To prevent this type of attack, the system should be able to evaluate of any command it receives to determine its validity.

Changes cannot be implemented in how the ViewPoint software issues commands or how the gimbal responds to them because they are commercial products and the source code is not available. However, it is possible to place a piece of in-line hardware or software on the UAV that receives the command packet before the gimbal and can decide whether or not to forward it along to the gimbal based on mission conditions.

Several methods can be used to make decisions on the validity of gimbal commands. One method to help catch unauthorized commands is to implement an authentication scheme, possibly by appending a cryptographic signature to messages sent from the ViewPoint software to the gimbal. However, this will not protect the gimbal system from the case in which a malicious agent has compromised the PCC. Depending on the degree of compromise, the malicious agent could still be able to send messages the UAV would consider authorized.

In a similar manner, providing additional authentication to commands capable of causing damaging effects would be helpful but not sufficient. An attacker who has not fully compromised PCC to the point of recovering the cryptographic key would be halted by such a defense, but further compromises may render this ineffective.

To protect from compromises of PCC the UAV should be able to judge the legitimacy of commands. To do this a run-time analysis can be performed to determine whether or not executing a command makes logical sense. For example, if the system received a command to retract the gimbal while it is in a pre-specified area of interest an intelligent decision would be to not immediately trust the command and attempt to verify its authenticity. In addition, authorized operators should be able to issue whatever commands they need, so there must be an override capability to verify that traditionally illogical commands are in fact legitimate.

3.3.2.3.1 Using Mission Context to Detect Gimbal Attacks

Cloud Cap software provides flexibility for a wide variety of mission operations, which makes the system susceptible to inside attacks involving seemingly valid commands that interfere with user operations. To prevent these, systematic rules based on mission context have been developed to limit when and where certain commands should be considered authentic.

The following algorithms use structures and methods from the software development kit provided by Cloud Cap for ViewPoint plugin creation and are aimed toward detecting the attacks found most feasible from section 3.2.3.3. Despite the following algorithm being written using an SDK, one could decode the information bytewise from the message streams and follow the same algorithms.

3.3.2.4 Packet Detection

The method LookForGimbalPacketInQueue() searches through a queue of packets and determines if a packet of gimbal type (i.e., a gimbal packet) is present in the message stream. It then stores this packet in a predefined buffer. The packet is then inspected to see if the packet type is a gimbal command. All of the vulnerabilities in section 3.2.3 fall into this type with the exception of the user warning packet.

3.3.2.5 Retract/Deploy Command Detection

The gimbal packets are further inspected to determine if the packet group is that of *Gimbal command and control group*. If so, then it is passed to the method that checks if it can be decoded into a retract/deploy struct pointer. If the method returns false the packet is ignored and the monitoring of the stream for packets continues. If the method returns true then the stream is decoded into information determining whether the gimbal is being commanded to either retract or deploy.

Under the assumptions that normal operations would entail the retraction and deployment of the gimbal directly after take-off and directly before landing, the velocity of the gimbal relative to Earth and the distance of the gimbal from the ground station should be relevant criteria to determine whether the gimbal retract/deploy command appears to be authentic.

The aircraft velocity and position can be determined by monitoring the gimbal telemetry stream for packets of type HOST_GPS_DATA_GIMBAL_PKTTYPE and of group GIMBAL_POSITION_INFORMATION_GROUP. These telemetry packets can be decoded to give the GPS position and velocity of the aircraft. These two pieces of information can be used to determine what phase of flight the aircraft is in. If the phase is take-off or approach/landing, then the retract/deploy command is considered authentic. If the aircraft is in cruise or loiter mode then the retract/deploy command should be considered malicious.

For the initial flight evaluations in Phase 2, it was determined that we should use an approach where if the aircraft was flying into a particular geographic region as determined by an adversary, that the camera would retract and not function whenever attempts were made to use the imagery collection system.

3.3.2.6 Erratic Gimbal Command Detection

To protect against a Gimbal Command attack it is assumed that during normal operations the gimbal should never be slewed to view a location above the horizon. Similar to the process in section 3.3.2.5, the telemetry stream is checked for gimbal packets in the queue. The method DecodeGimbalCmdPacket() is used to give an elevation angle of the gimbal. Gathering the GPS information using the same algorithm in section 3.3.2.4, the aircraft altitude is determined. If the aircraft altitude is higher than the altitude at which the gimbal is pointed, then the command is authentic. If the gimbal is pointed at a higher altitude than the aircraft then the command is considered malicious and the user can be warned via a message sent through a payload message stream using the autopilot command and control link.

Further constraints can be placed on the gimbal angles by limiting the gimbal orientation based on mission CONOPS. For example, if the UAV mission is to loiter overhead a specified target then the gimbal field of view should never extend outside the orbit of the aircraft.

3.3.3 Hardware Security Against Design and Manufacturing Attacks – Securing the Protocol Conversion

3.3.3.1 An FPGA-Based Application of Protocol Conversion with TMR

Today, many mission-critical and safety-critical systems, including the target UAV system of this project, depend on integrated circuits (ICs). The supply chain for integrated circuits (ICs) is difficult to secure, involving multiple firms (processor design, fabrication, packaging, etc.), each with large teams of designers and technicians. A “hardware Trojan” can be very small and easy for a single individual to insert, especially at the design stage, where it might be as simple as one or two lines of innocuous-looking programming, or a few transistors and wires in a physical chip layout. Consequently, effective security techniques are required to prevent adversaries from interfering with the correct operations of the UAV system.

Our solution for this project is triple modular redundancy (TMR). The three components vote on all actions, ensuring that a single corrupt unit will be outvoted. Redundancy also provides fault tolerance at no extra cost, and fault tolerance is becoming increasingly important as transistor miniaturization continues and transistors become more vulnerable to electrical upsets.

Compared to software-based protections, which usually have a "big picture" of what the system is doing on the function level, hardware-based protections focus on behavior on the instruction and word level, and require very high level of hardware sophistication and coordination to defeat, hence increase the difficulty to attack. Our TMR solution operates at the hardware layer and directly protects the most critical system functions. These solutions are embedded within the protected hardware and don't require modification to the application software.

The target UAV has an autopilot system and a monitoring system, called Piccolo and Sentinel, respectively. Piccolo communicates using RS-232 protocol, while the Sentinel communicates using TCP/IP protocol. We first implement an application with the basic functions: 1) protocol conversion between RS-232 and TCP/IP, and 2) SD card access. These two functions are the initial test cases in this project. Then TMR is applied to protect the critical hardware, e.g. the microprocessor, in the protocol converter.

The re-configurability and flexibility of field programmable gate array (FPGA) makes it favorable for the purpose of prototyping and proving our protection concepts. In this report, the soft-core processors for protocol conversion and all hardware-based protections are implemented on FPGAs to verify their feasibility. In addition, FPGAs may also be suitable for the purpose of deployment, because of their short design-to-product time.

The following subsections will describe the functionality of the initial test cases.

3.3.3.2 Protocol Conversion

Serial input data in RS-232 protocol is converted to TCP/IP packets and then sent through the Ethernet PHY. TCP/IP input packets are analyzed and the data bytes are converted to RS-232 format, and then sent through the universal asynchronous receiver/transmitter (UART) pin.

The TCP/IP stack needs to process two data streams: one for Piccolo data and the other for user data. Each stream occupies an individual port of the TCP/IP stack.

3.3.3.3 SD Card Access

Besides protocol conversion, the application also needs to store in the SD card the gold-standard waypoints information sent from the Sentinel, and read it when requested. In order to protect the data stored in the SD card, Triple-DES encryption algorithm is applied to both write and read access of the SD card.

3.3.3.4 Design Implementation

The application is implemented in a custom board, SiCore SHIELD II board. Below is a summary of the board (Digilent, 2013):

- Xilinx Kintex-7 XC7K325T-1FFG676 FPGA
- Low-jitter 200 MHz oscillator
- Four 10M/100M/1G Ethernet PHYs with RGMII
- 1Gb BPI Flash
- One SD card slot
- Eight UART transceivers
- Four on-board LEDs and four on-board general-purpose buttons
- 512MB DDR3 memory (800 MHz)

Figure 38 shows a picture of the SiCore SHIELD II board.

Further Modifications to SiCore SHIELD card for the Sentinel creating the SHIELD 2 card

- Removal of the PPC460EXr

  Since bitstream integrity is handled by the FPGA, the PPC460EXr was removed from the UAV SHIELD. The cryptographic capabilities are handled by the FPGA. Interfaces to the RS-232 and Ethernet ports are also handled by the FPGA.
- Removal of the MAXQ1103 and Anti-Tamper Circuitry

  The anti-tamper circuitry was removed to reduce the weight of the card. With this removal, it also allowed the removal of the Cyclone II FPGA, which acted as a conduit between the PPC and MAXQ and battery holders which were used for backing up the MAXQ's battery-backed and zeroizable memory.
- Removal of the PCIE Interface

  The UAV SHIELD operates as a standalone card and does not interface with a host system, which allowed the PCIE interface to be removed.
- Switch from SATA HDD to Secure Digital (SD) Card

The amount of storage offered by a SATA HDD was not needed for the UAV SHIELD. In addition, a change to an SD card reduced the weight of the card and the number of components with moving parts.

- Addition of RS-232 and Ethernet Interfaces

  The UAV SHIELD communicates with three Raspberry Pis through Ethernet, which required the addition of more Ethernet ports. It communicates with other hardware on the UAV, which required the addition of an RS-232 octal UART chip and eight RS-232 ports.
- Additional Modifications
  - 512MB DDR Memory upgraded to 1GB
  - 64MB Flash Memory upgraded to 128MB

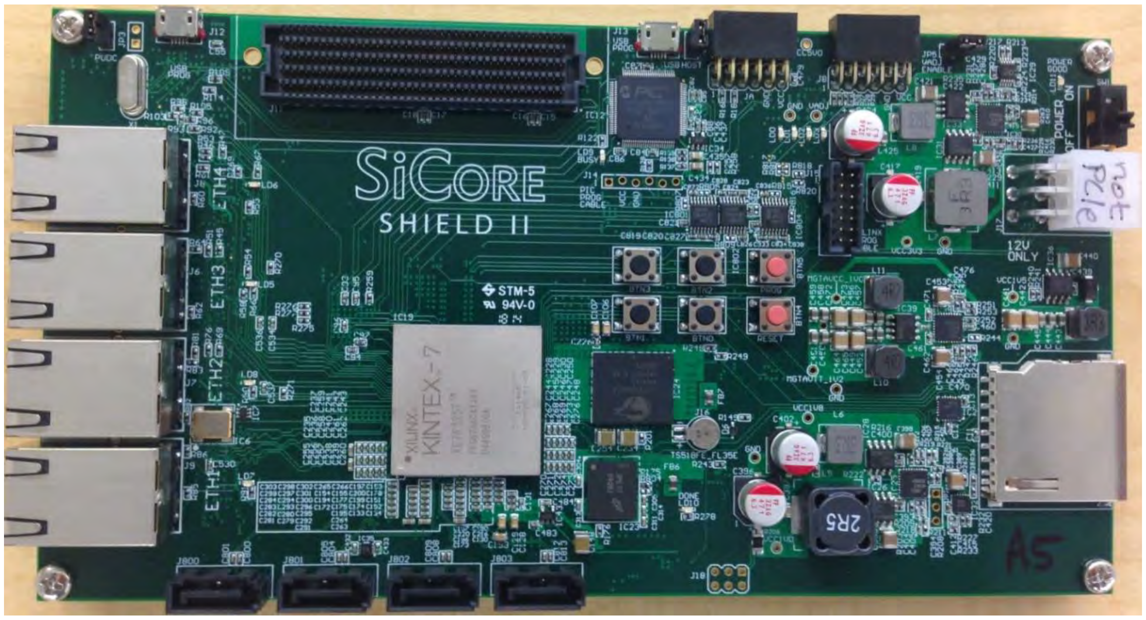


Figure 38 - SiCore SHIELD II Board

We choose an FPGA to implement our design due to two reasons: (a) FPGA is great for the purpose of prototyping because of its re-programmability and fast reconfiguration time, and (b) there are multiple open-source soft-core processors available for FPGA implementation.

We choose to build the protocol conversion and SD card access functionality on LEON3, an open-source soft-core processor (LEON3 Processor). LEON3 uses SPARC V8 instruction set and AMBA-2.0[2] AHB bus interface. It is released as synthesizable VHDL files, and is configurable through the use of VHDL generics.

The protocol conversion and SD card access is implemented as a bare-metal application, meaning that there is no operating system running on the processor. Less complexity in the

[2] AMBA is a registered trademark of ARM Ltd.

implementation means less design effort and more importantly, fewer loopholes that might be taken advantage of by adversaries.

Figure 39 shows a simplified block diagram of the LEON3 architecture.

Due to the lightweight use of memory in this application, we use on-chip block RAMs (BRAMs) to implement the RAM and ROM. The BRAMs are memory resources available in the FPGA chip, and enables faster access time. The ROM is a module written in VHDL that contains hard-coded instructions, which are automatically loaded to the CPU when the FPGA configuration is complete. The ROM can be generated from C program using a PROM generator (see Section 3.1 for more details). AHB bus is used for high-speed operations, such as writing and reading data between CPU and memory. APB bus is used for low speed operations, such as communications to the peripheral components. The peripheral components include general purpose I/O (GPIO), UART, Ethernet PHY, and etc.

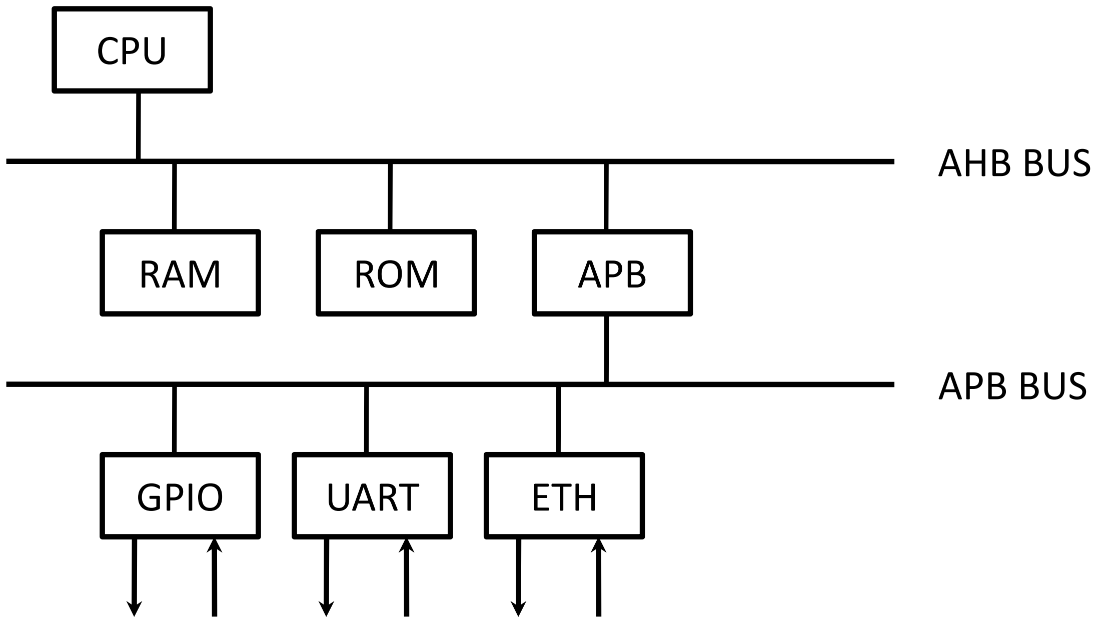


Figure 39 A Block Diagram of the LEON3 Architecture

3.3.3.5 Protocol Conversion

We use uIP, an open-source TCP stack, to establish TCP/IP connections and process TCP/IP packets. Figure 40 shows the implementation of the TCP stack.

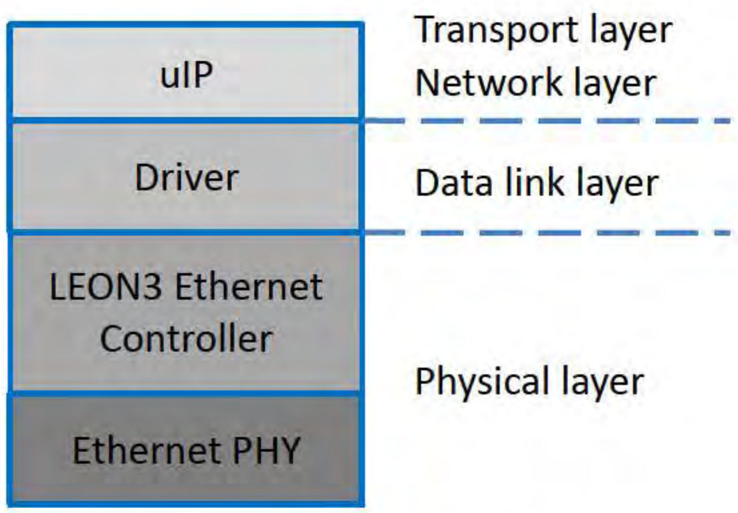


Figure 40 Implementation of TCP Stack

On the physical layer, LEON3 Ethernet controller drives the pins of the Ethernet PHY. On the data link layer, a driver writes data to the transmitter of the Ethernet controller and reads data from the receiver. On the network layer and transport layer, uIP establishes and manages TCP/IP connections, e.g., sending and receiving packets, updating acknowledgement number and sequence number, etc.

The received TCP/IP packets are first analyzed, and then the data bytes are sent to the buffer of the UART transmitter. When the DATA_READY flag of the transmitter's buffer is high, the controller reads from the buffer and sends the data in RS-232 format. When there is incoming serial data, the DATA_READY flag of the receiver's buffer is set high, and the data will be processed when the CPU is idle.

3.3.3.6 SD Card Access

FatFS, an open-source FAT file system (FatFS - Generic FAT File System Module), is used to implement the SD card access functionality. FatFS uses GPIO signals to drive the pins of the SD card slot in serial peripheral interface (SPI) mode. Write and read commands can be sent to the SiCore board through the user TCP/IP stream.

To issue a write command to the SD card, the user can send the following text to the TCP/IP port of user data:

```
Write SD <content>
```

The command is case sensitive. After receiving the write command, the program will invoke the SD write function to write <content> to SD card.

To issue a read command to the SD card, the user can send the following text to the user data port:

```
Read SD
```

The command is case sensitive. After receiving the read command, the program will invoke the SD read function to read from the SD card.

In order to protect the data stored in the SD card, Triple-DES encryption algorithm is applied in the design. We choose an open-source hardware module (3DES (Triple DES) / DES (VHDL) :: Overview) to implement it. The use of hardware encryption can make the encryption process faster as well as reduce the software code size. When a write command is issued, the data is first sent to the Triple-DES module for encryption. The encrypted data is then written to the SD card. When a read command is issued, the encrypted data is read from the SD card and then sent to the Triple-DES module for decryption.

3.3.3.7 TMR – Triple Modular Redundancy

The entire LEON3 implementation described earlier, including CPU, buses, RAM, ROM, and all peripheral components, is triplicated in the FPGA. The three implementations currently have the same configuration, share the same clock, reset, and input signals. Additional comparators and MUXs are created to act as majority voters. The output signals of GPIO, UART, and ETH components are sent to their individual voters.

The most important function that TMR performs is to compare the output signals of the three implementations. In order to compare the UART output data, three FIFOs are used to store the data sent to the UART transmitters, one for each implementation. Every time data is sent from the bus to the UART transmitter, a copy of the data is also written to the corresponding FIFO. When all FIFOs are written, the three copies of data are read and compared. Similar approaches are applied to compare the output signals of GPIO and ETH components.

Figure 41 shows a block diagram of the TMR design and its connections to other systems.

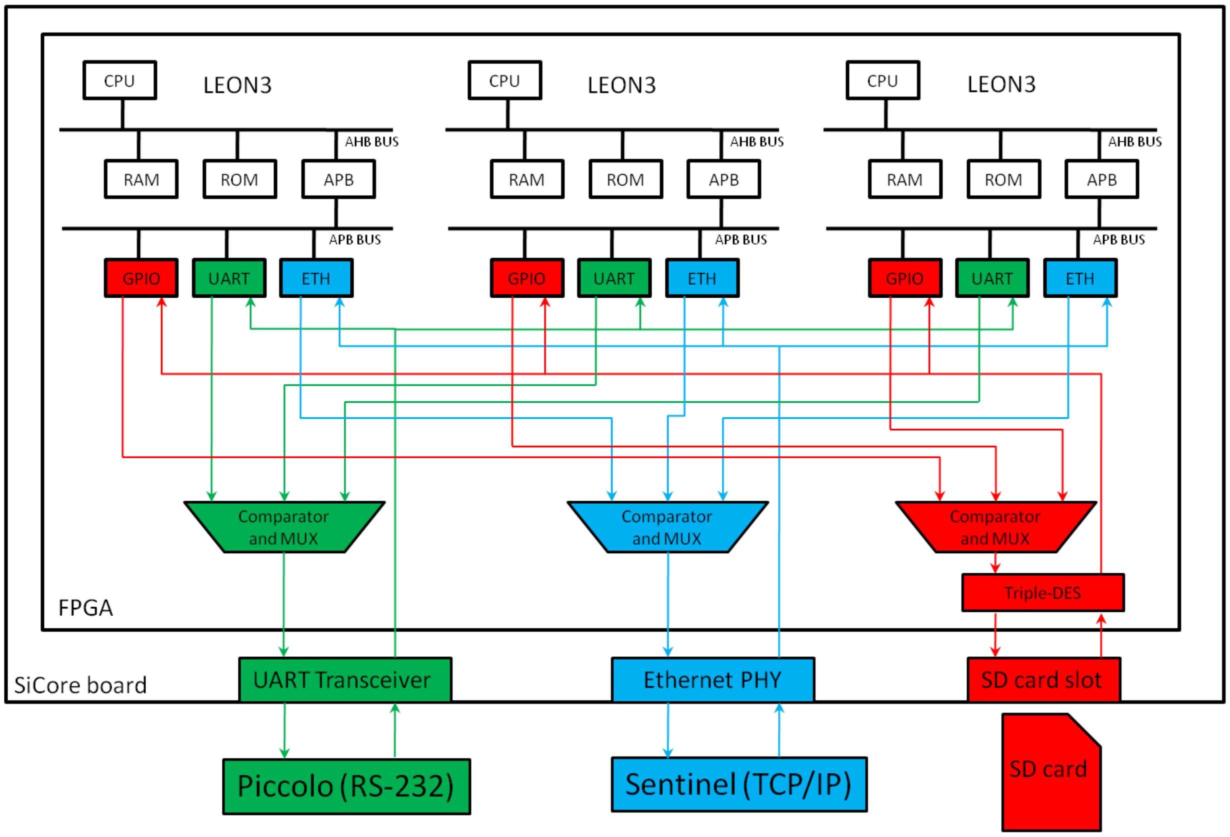


Figure 41 **A Block Diagram of the Protocol Conversion Application with TMR**

In a TMR design, synchronization between the three implementations is critical to ensure correct operation. A small difference in the progress of program execution can result in different outputs, which will lead to false positives reported by the TMR design. Since the three LEON3 implementations have identical configuration and share the input signals, ideally they should have exactly the same behavior in a cycle-by-cycle fashion.

In practice, however, we observed out-of-sync behavior among the three implementations. The cause is still under investigation. In order to make the TMR design work correctly, we added several synchronization points in the program to enforce synchronization. When a synchronization point is reached in program execution, a GPIO sync_out signal is set to 1 and this signal is sent to the top level of the TMR design. When all three sync_out signals are 1, a controller sets a GPIO synchronize signal to 1 and this signal is sent back to each LEON3 implementation. After seeing the asserted synchronization signal, each LEON3 implementation sets their own sync_out signal to 0 and proceeds in program execution.

This synchronization method has been proved to be reliable by running tests for tens of hours without errors.

3.3.3.8 Lessons Learned

This section summarizes the problems and issues we have encountered, as well as the solutions we took during the implementation of the design.

3.3.3.8.1 Generating VHDL PROM Module from C Code

Although we can use a JTAG cable to load an executable to the LEON3 core, it is more convenient and practical to add a PROM in the design so that the LEON3 core can automatically load program from it when the FPGA configuration is done.

To generate a VHDL PROM module, the following commands are used:

```
sparc-elf-gcc uip_sd_sicore_tmr.c -c -o uip_sd_sicore.o -luip

sparc-elf-gcc sd.c device.c ff.c mmcbb.c uip_sd_sicore.o libuip.a -o uip_sd_sicore.exe

mkprom2 uip_sd_sicore.exe -freq 50 -msoft-float -baud 57600 -ramsize 128 -romsize 1024 -romws 15 -
romwidth 64 -rmw -dump -o uip_sd_sicore.out

make ahbrom.vhd FILE=uip_sd_sicore.out
```

The first and second commands compile the C code and generate an executable. The third command takes the executable and generates a .out file. More details of mkprom2 options can be found in (Eisele, 2013). The last commands takes the .out file and converts it to a VHDL module. By instantiating the ahbrom.vhd module, the PROM is added in the design.

A lesson we have learned in generating the PROM is that we need to keep the file size minimum. Our initial approach to generate the PROM was as follows: 1) Compile all .c files and generate separate .o files. 2) Use all .o files to generate the .out file. 3) Use the .out file to generate the VHDL module. The drawback of this approach is that the generated .out file is considerably large and results in a large PROM file (>1MB), which is not able to fit in the FPGA BRAMs. As a result, the PROM is mapped to LUT RAMs in the implementation phase, drastically increasing the used logic and routing time. The new approach presented above (.c -> .o -> .exe -> .out -> .vhd) keeps the generated PROM at an acceptable size (~600KB) so that it can be implemented using BRAM primitives in the FPGA. This greatly reduces logic use and place and route time during the implementation phase.

3.3.3.8.2 Sending Packets in uIP

The TCP stack we use, uIP, is designed to process only one outstanding TCP packet at a time, meaning that an ACK must be received before uIP can send another packet. Therefore three states are added in the application code to control the sending status:

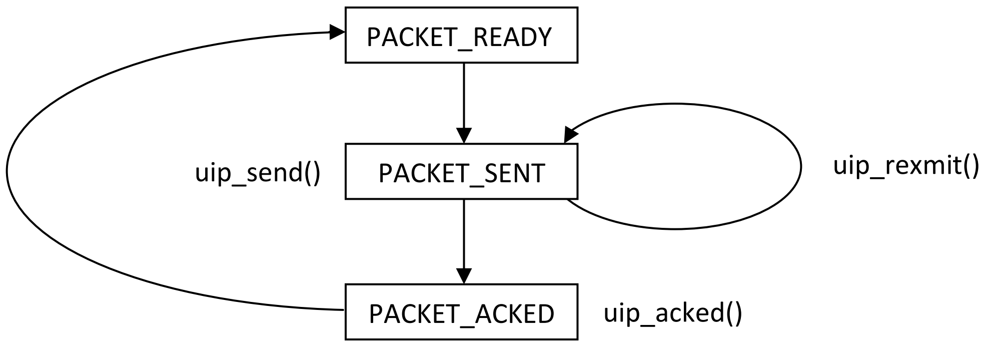


Figure 42 States of Sending TCP Packets in uIP

As illustrated in **Figure 42**, there are three states in sending TCP packets using uIP: PACKET_READY, PACKET_SENT, and PACKET_ACKED. After a packet is sent, the application code enters the PACKET_SENT state. If the timer expires, then function uip_rexmit() is called for a retransmission. When an ACK for an outstanding packet is received, the application code enters the PACKET_ACKED state. Then if a new packet is ready to be sent, the code enters the PACKET_READY state. The code will only enters the PACKET_READY state when it is already in the PACKET_ACKED state and new packet is ready. In this way, the order of receiving ACK and sending new packet is ensured.

3.3.3.8.3 TCP Delayed ACK

TCP delayed acknowledgement is a technique used by some TCP/IP implementations to improve network performance (TCP delayed acknowledgement). Several ACK responses may be combined together into a single response to reduce the overhead in transmission. A TCP receiver using this algorithm will only send an ACK for every other received packets (Dunkels). After receiving a packet, if no following packet is received within a specific time period, then an ACK is sent. In Windows Vista and 7, this time period is set to 200ms.

As mentioned earlier, uIP is designed to process only one outstanding TCP packet at a time. Therefore, the feature of TCP delayed ACK practically reduces network performance as each transmission has to wait for at least 200ms.

To solve this problem, we can set two Windows registry keys, TcpAckFrequency and TcpNoDelay, to 1. By setting these two registry keys, we force Windows to send an ACK for each packet it receives.

Linux implementations should have similar settings. If the TCP throughput is lower than expected, then these settings should be checked.

3.3.3.8.4 Bug in uIP

The uIP version we use, uIP 0.9, has a bug in the code for updating TCP/IP sequence number. In a rare case where uIP processes a new received packet immediately after it sends one, it wrongly sets the length of outstanding data to 0. In this case, uIP "forgets" that it still has outstanding data that has not been acknowledged, and the following sequence numbers will be wrong.

This bug has been fixed. A new release of uIP, uIP 1.0, has also fixed the bug and reported it in its log file.

3.3.3.8.5 Bug in ETH driver

Gaisler provides a driver for uIP to run on LEON3 Ethernet PHY, but it has a bug which reduces performance. In uip_greth.c, function libio_uip_greth_initialize_hardware ():

```
GRETH_REGSAVE(rx_bd[i].stat, GRETH_BD_EN);
```

should be changed to:

```
GRETH_REGSAVE(rx_bd[i].stat, 0);
```

In uip_greth.h, GRETH_BD_EN is defined as 0x800, which corresponds to the enable bit of the receive descriptor (Gaisler, 2013). The enable bit being 1 indicates that the receive descriptor is busy and other accesses must wait until the bit is 0. The original line sets the enable bit to 1 during hardware initialization, and therefore unnecessarily postpones the access to the receive descriptor (because the program has to wait until enable bit is 0). The modified line fixes this problem and significantly improves the speed in TCP packet receiving.

3.3.3.8.6 UART Flow Control Signals

In RS-232 protocol, optional RTS (ready to send) and CTS (clear to send) signals are used to control the data flow between a transmitter and a receiver that are at different speeds. The UART transceivers on the SiCore board do not include these signals. In the LEON3 implementation, the VHDL generic "flow" should be set to 0 (default is 1).

3.3.3.8.7 Clock Signal for Ethernet PHY

The Ethernet PHY works in 10Mb full duplex mode, which requires a 2.5MHz clock signal for the PHY. In Kintex-7 FPGAs, the minimum frequency an MMCM (mixed mode clock manager) can generate is 10Mhz, therefore the PHY clock signal cannot be generated using MMCM. Instead we used a counter to generate the required 2.5MHz clock signal.

Note that the user generated clock signal should be routed using dedicated clock trees instead of general routing paths to minimize clock skew. This can be done by sending the clock signal to a BUFG primitive, which is part of the FPGA clock trees.

3.3.3.8.8 Pin Assignments for LDs

We started the project with the CML NetFPGA board and later switched to the SiCore SHIELD II board. The two boards have almost the same components and configurations, but there are some differences that should be noted.

The FPGA pin assignments for LDs are different on the CML NetFPGA board and the SiCore SHIELD II board:

	CML NetFPGA	SiCore SHIELD II
LD0	E17	AF14
LD1	AF14	W19
LD2	F17	E17
LD3	W19	F17

Table 11 Different Pin Assignments on NetFPGA and SiCore Board

When switching to other boards, the designer needs to pay attention to the differences in pin assignments. These signals include Ethernet PHY signals, UART signals, buttons, LDs, and clock signals. When targeting the design to other FPGAs, the design needs to be modified to suit the new FPGA. Gaisler provides template LEON3 designs for various FPGAs.

3.3.3.8.9 Buffer Size of UART Transceiver

The original UART module in LEON3 design has buffers for its transmitter and receiver. The buffer size is configurable from 0 to 32, which is far from enough for this application. When the CPU is processing other tasks, for example, sending TCP packets or reading from SD card, serial data may come in simultaneously and cause overflow. Therefore larger buffers are required for correct UART operations.

The original UART buffers are implemented using LUTs (look-up tables), which are distributed and expensive FPGA general logic resources. We replaced them with more centralized BRAM FIFOs. Each transmitter and receiver has a 4KB buffer, which has been proven by tests to be sufficient.

3.3.3.8.10 Timing Requirements

The target frequency of this design is 100MHz. To achieve this frequency, some unnecessary components in the LEON3 design can be removed to improve the delay in the critical path. Several settings are changed in the configuration file config.h:

```
constant CFG_LDDEL : integer := 2;

constant CFG_MMUEN : integer := 0;
```

3.3.3.8.11 Synchronization

TMR synchronization has brought problems to the design. We have observed out-of-sync behaviors among the three LEON3 implementations. Different progress in program execution could result in different behaviors and cause TMR to report false positives when there is no attack. The source of the diverged timing behavior is still under investigation. Possible reasons include:

- The LUT-generated clock signal for Ethernet PHY has glitches
- Clock signal has skew in distribution

For example, the following code reads data from the UART receiver buffer:

```
while(uart_1[1] & DATA_READY) {

        uart_1_in_buf[uart_1_in_len++] = uart_1[0];
        if(uart_1_in_len == 1446)  break;    // Max length reached
    }
```

The while loop checks the status register of the UART receiver to see if it has data ready to be read. Keep in mind there are three copies of this code running simultaneously. It is possible that the execution of copy 0 is a little ahead of copy 1 and copy 2, and when it reaches the while condition line, no more data is ready, therefore it exits the while loop. Then copy 1 and copy 2 reaches the while condition line, and a byte of new data has just arrived in the small gap. In this case, copy 1 and copy 2 will continue execution in the while loop to read this new byte. As a result, the data held by copy 0 is different from that of copy 1 and copy 2, and a mismatch in behavior will be reported later by TMR.

A solution is to wait for a while after the DATA_READY flag becomes low. In the following modified code, a new line, delay_us(174), adds a 174ms delay, which is the time for receiving one character with Baud rate 57600. In this way, the faster copy (copy 0) will wait for another character to come. If no character is received during this time, then it means that this chunk of characters is over, and it is safe to exit the while loop.

```
    do {
        while(uart_1[1] & DATA_READY) {
            uart_1_in_buf[uart_1_in_len++] = uart_1[0];
            if(uart_1_in_len == 1446)  break;
        }
        delay_us(174);  // wait for another character, baud rate = 57600
    } while (uart_1[1] & DATA_READY);
```

Similarly, when three copies of the same program reach an if condition, a mismatch could occur. For example:

```
    // Read UART input when the previous packet has been acked
    if((uart_0[1] & DATA_READY) && (stream_0_state == PACKET_ACKED))
        uart_0_input();
```

Again, if copy 0 is faster than the other two, then it is possible that copy 0 judges the if condition as false, while the other two judge it as true. In this case, copy 0 will not read the UART input, but copy 1 and copy 2 will. As a result, copy 0 does not have TCP packet to be sent, whereas copy 1 and copy 2 will send as TCP packet the data they read from UART. This again will cause a mismatch in behavior that will be reported by TMR.

To solve this problem, synchronization points are added into the code as described in Section 2. An example of the modified code is shown below. The added synchronization point ensures that the three copies of code judge the if condition at the same time.

```
synchronize();

// Read UART input when the previous packet has been acked

if((uart_0[1] & DATA_READY) && (stream_0_state == PACKET_ACKED))

        uart_0_input();
```

Figure 43 shows a waveform when a synchronization point is reached. Signals sync_out_0, sync_out_1, and sync_out_2 are outputs from the GPIO components. They are asserted to 1 to indicate that the three copies have all reached the synchronization point. When all three signals are asserted, signal synchronize, an input to the GPIO components, is asserted as a response. After seeing the asserted synchronize signal, each copy de-asserts its sync_out signal, and proceeds in execution. After sync_out signals are de-asserted, synchronize is also de-asserted.

The extra synchronization points lead to performance overhead to the TMR design, because the faster copies need to wait for the slower ones when they are out-of-sync. It is observed that the gap between the fastest and the slowest copies can reach 40 clock cycles, or 400 ns when the clock frequency is 100 MHz. This amount of delay has barely any impact on real-time operations of the UAV. Since the three copies are periodically forced to synchronize, the gap in delay will not accumulate over time.

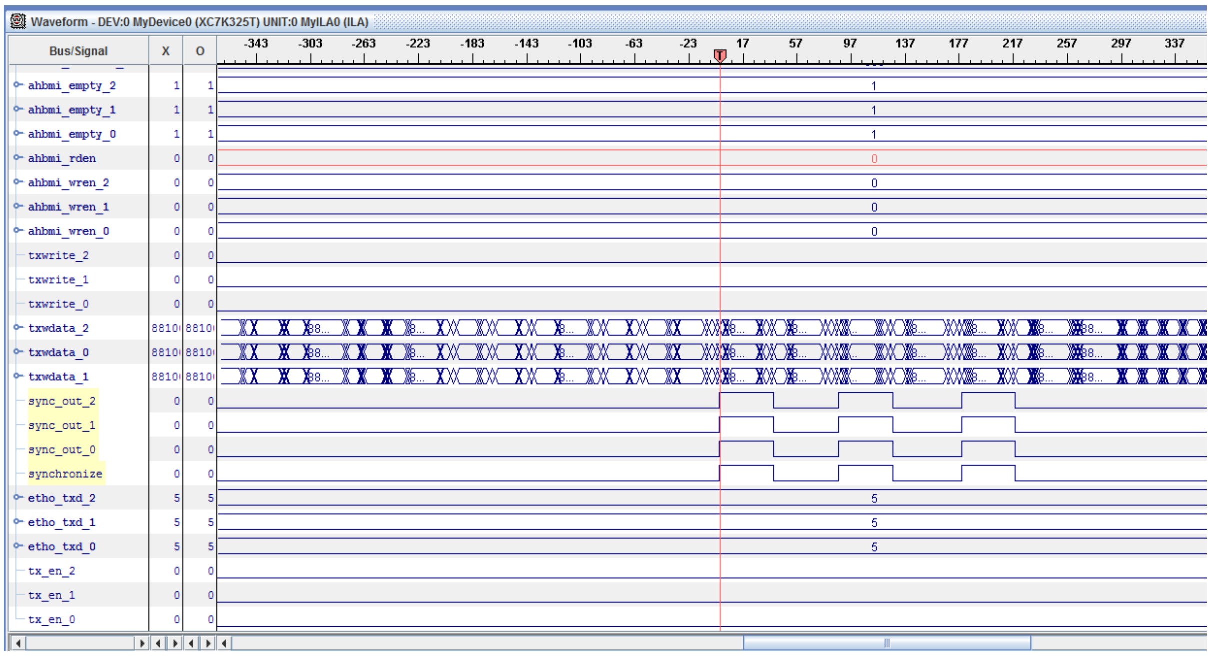


Figure 43 Waveform of Synchronization Point

3.3.3.8.12 Solutions and Detection/Restoration Mechanisms

Detection of malicious or faulty operations is usually performed by adding hardware redundancy, such as dual modular redundancy (DMR) and triple modular redundancy (TMR). With DMR recovery can be done by periodically storing circuit states into checkpoints. When a mismatch between the two copies is detected the latest checkpoint is used to restore the system to a correct state. With TMR a fault can be masked by using a majority vote. The output of the voter can also be fed back to all three copies so that they all keep the correct value.

In addition, hardware redundancy can provide fault tolerance. Radiation-induced single event upset (SEU) can cause transient errors in electronic systems, and the error rate increases as the altitude rises. As technology node shrinks it is more and more likely that an SEU causes multiple bit errors in a memory cell. Therefore it is important for aircraft, such as UAVs, to be fault tolerant. When redundancy is applied to a UAV for the purpose of security it also brings the ability of fault tolerance for free.

Assuming that a processor could be compromised by supply chain attacks, it is insecure to use three identical processors from the same source. Thus, heterogeneous cores from different manufacturers are considered. Processors with different instruction set architectures (ISA) are an option, but different ISAs may result in different instruction and/or memory access orders, which significantly increase the difficulty of synchronization. Unlike software synchronization, which can be done by inserting synchronization points into the code, hardware synchronization can only be done by monitoring register and memory values. Therefore heterogeneous processors with the same ISA are more favorable. The processors can have different configurations such as speed, clock signal, cache size, and etc.

Similar to synchronization, detection and restoration of deviated operations also depend on monitoring register and memory values. The assumption is that if multiple processors perform exactly the same operations (e.g., write the same data to memory) on an instruction-by-instruction basis then they are considered to have the same behavior.

3.4 Evaluative Criteria

As the project has progressed from the formulation of System-Aware security patterns to a prototyping pilot effort used for validating the System-Aware cyber security concept, our team has been addressing a set of generic design-related questions that can support future efforts related to implementations of the System Aware Cyber Security concept. These questions continue to be addressed in the form of additional use cases for other systems that can potentially benefit from the System-Aware techniques. The answers to these questions impact the potential viability of using the System Aware concept in a potential application and the level of performance that can be achieved:

- **What are potential attacks?** Which system components and functions are most critical to the system? How vulnerable to attack are they, and how could an adversary do the most damage to degrade functionality with the least cost to the adversary? In turn, which attacks can we protect against for the least cost to us while increasing the cost and complexity to the adversary?
- **What are the available data measurements from the system to be monitored?** In order to provide a reliable Sentinel platform to detect and classify anomalous behaviors and attacks in

critical functional areas, we must possess the ability to interface with and to extract data from those critical functions. In addition, the data that can be extracted may directly affect the security design patterns that are employed to enhance a given system's security. For example, as the variety of measurements about the state of a critical function increases, so does the potential number of diversely redundant algorithms available for ensuring the integrity of that the critical function. Finally, the amount of data that can be extracted from the Sentinel is critical to accurately gauge the Sentinel's ability to protect a system function and restore that function when it is under attack.

- **What should be measured to protect against potential cyber-attacks?** What are the critical pieces of information that are needed to adequately determine the state of the system? Should new sensors be added to the system to enhance the monitoring capabilities of the Sentinel while not degrading normal system behaviors? If information is needed from multiple parts of the system to verify a system state, how is that information integrated to provide an accurate system state and better detection of anomalous system behavior?
- **Can we standardize the data collection protocols for collecting the data provided by the monitored system?** The ability to standardize the data extracted from system functions into a form that can be utilized by the Sentinel is necessary to integrate with legacy systems and facilitate reusability across a diverse set of domains. This standardization can potentially impact the Sentinel's ability to deal with the timing and latency issues associated with monitoring functions, differing interfaces for the system that are required to extract the data, and the potential collateral effects on the system function being monitored and on other parts of the system.
- **What is the rate of the data measurements that are needed to adequately detect a cyber-attack?** To understand this question, one must investigate the normal rate of change of the system configuration, the nature of specific attacks, the rate of change in system configuration that would be deemed to be unacceptable, the consequences of potential attacks, acceptable responses to successful exploits, the stability of the configurations of the system, and the sensitivity of the rate of change of those configurations related to the monitoring and detection functions of the Sentinel.
- **What are the methods needed for assuring the integrity of an operation?** When looking at the critical system functions, which security design patterns make the most impact in providing protection for the system without hindering the operation of the system? Which patterns create the greatest difficulty for adversaries in terms of developing alternative attacks that achieve similar outcomes? If you distribute those security design patterns across several platforms, how do they communicate and how often do they update each other?
- **What is the complexity of the algorithms used for securing the system to be protected?** We must evaluate the complexity of the algorithms and the tradeoffs with complexity versus system security and system performance.
- **How should the Sentinel respond once an attack has been detected?** Under what circumstances does the system automatically get restored to another state? If the system is not automatically restored, who should be informed in the event an attack has taken place? What

information should those individuals be provided, and what options are they given for restoration? Should the attack be allowed to continue for analysis purposes so as not to tip off the attacker that their attack has been detected?

As we continue to address new types of physical systems where there will be potentially more time sensitive systems to protect and also the introduction of multiple Sentinels in place to protect those functions, this framework of questions will continue to be refined.

4 Phase II - Flight Evaluations

This section outlines the Phase 2 efforts by the UVa and the GTRI to transition the System-Aware cyber security solutions developed under RT-42 into a Sentinel configured to meet the size, weight, power and functional requirements necessary for airborne use, including a flight-ready demonstration of the Sentinel. As part of this effort, a flightworthy Sentinel has been developed based on the SiCore SHIELD II CoProcessor board discussed in section 3.3.1.2. The Sentinel algorithms for detecting the attack, alerting the operator, and taking remedial action will be based upon those developed in RT-42 have been developed on diverse Single Board Compute (SBC) platforms. The following subsections describe the tasks that were performed in order to prepare and conduct flight evaluations of the Sentinel concept.

4.1 Flight Evaluation Results Overview

As part of the ongoing research effort described in Phase 1, a prototype Sentinel system was designed, developed and deployed on a Griffon Aerospace MQM-170 Outlaw UAV equipped with a Cloud Cap Piccolo II autopilot and Cloud Cap TASE Gimbal video surveillance payload. The Piccolo II autopilot includes a pilot station to issue commands to the aircraft, as well as switch between autonomous and manual control. It also includes a ground station to transmit and receive information from the aircraft to the pilot station. The deployment of the prototype Sentinel included conducting flight evaluations at the Early Count Airport in Blakely Georgia. The flight evaluations were designed to emulate the scenario in which a UAV that is unable to accomplish its assigned surveillance mission to identify and track targets due to a cyber-attack embedded into the UAV’s autopilot. Figure 26 shows the flight plans used as part of the flight evaluation activity.

The flight evaluations included 2 flights of the UAV to test the Sentinels ability to protect against 3 different cyber-attack scenarios

1. *Cyber-Attack against the UAV’s payload*: This scenario focused on disrupting the UAV’s ability to gather video surveillance of an adversary’s facility—represented by a hangar located at the airfield—by disrupting the onboard video surveillance system when it entered a pre-determined geographic area. This attack was initiated by a cyber-attack box deployed on the aircraft to emulate a cyber-attack embedded in the autopilot responsible for relaying a remote pilot’s commands to the UAV’s payload. This attack included two levels of disruption:

    a. *Prevent the collection of all video surveillance data in a specified geographic region*: Whenever the UAV entered a pre-determined geographic region the cyber-attack would automatically trigger the onboard camera system to retract inside the aircraft. In addition, as long the aircraft remained in the pre-determined geographic region, the cyber-attack would prevent the camera from redeploying.

b. *Prevent the UAV from tracking identified targets in a specified geographic region*: Part of the onboard camera system's capabilities includes allowing an operator to select a target for automatic tracking; i.e., the camera will automatically readjust the camera to keep the operator specified target centered in the camera's field of view. This cyber-attack focused on disrupting the camera's ability to track targets by either (a) retracting the camera ONLY when an operator issued a command to the onboard camera to track a target or (b) point the camera in a random direction away from any target the operator selected to track.

2. *Cyber-attack to corrupt the metadata associated with video surveillance*: This is the same scenario as described in Section 3.2.2.4; however, rather than preventing the onboard camera system from recording video, this cyber-attack would focus on reducing the value of the recorded footage by corrupting the metadata associated with the onboard camera's geographic location. Specifically, when the UAV entered the pre-determined Attack Active Zone, the cyber-attack would shift the latitude and longitude associated with the camera's geographic location by 1000 meters. In addition, to prevent an operator from noticing a sudden shift in the cameras reported position, the cyber-attack would shift the latitude and longitude by walking the UAV off at a rate of 100 meters per second. The cyber-attack was emulated by a cyber-attack box that would monitor and corrupt the GPS position reported by the autopilot to the onboard payload.

3. *Cyber-attack to corrupt the flight plan (parameters) of the aircraft:* This scenario is meant to closely emulate the cyber-attack scenario described in section 3.2.1.4. In this scenario the aircraft would be prevented from collecting surveillance data from a given geographic region by altering the aircraft's flight plan to redirect it away from a pre-determined region. To emulate this attack, a cyber-attack box was situated onboard the aircraft that would redirect the aircraft between two different flight plans. The flight plans selected were 0->1->2->3->0 and 10->11->12->13->10 as shown in Figure 26. In addition, for safety, the attack was initiated by an operator on the ground to ensure the aircraft would remain close to the airfield and minimize the turning radius needed for the aircraft to shift between flight plans. Finally, this attack included the ability to mask the pilot's operator display; i.e., the pilot's display would NOT show the aircraft moving between flight plans. Instead it would show the aircraft continuing along the original flight plan unaltered. This masking of the display was accomplished by corrupting the data reported by the ground station to the pilot display. For safety, the masking could be run independently from the attack onboard the aircraft.

All of the previously outlined scenarios were conducted at an altitude of 750 feet AGL.

To protect against the previously outlined scenarios two prototype Sentinels were used. In addition, a special cyber security operator station was created that could receive security information from the Sentinel, adjust the sensitivity of the Sentinel's detection algorithms, and allow an operator to use the Sentinel to take corrective actions to restore the system to a non-compromised state.

One prototype Sentinel was a payload deployed on the UAV with the following protections:

- Monitor the parameters of the UAV's autopilot to detect changes.
- Detect possible cyber-attacks against the UAV and report this to the cybersecurity operator located on the ground.
- Defend against cyber-attacks aimed at compromising the UAV's payload. This protection was accomplished by inserting a small microcontroller between the autopilot and the payload to monitor communications.
- Enable the cybersecurity operator on the ground to take corrective actions to restore the UAV to a non-compromised state after a possible cyber-attack has been detected.
- Designed with protections to secure the prototype Sentinel:
    - Three separate single board computers to vote on and perform all Sentinel monitoring, detections, and restoration actions. Only actions that passed a majority vote (2 out of 3) were accepted. All disagreements between boards were recorded for future forensic analysis and the cybersecurity operator on the ground was alerted immediately.
    - Diverse implementations of the cybersecurity algorithms. The algorithms providing protections against cyber-attack scenario 2 were implemented in two different programming languages: C and Python.
    - Isolated storage area containing the critical autopilot parameters of the aircraft before a flight is initiated. This was used to store the flight plans to protect against cyber-attack scenario 3.
    - Secure monitoring of the autopilot. Autopilot data monitored by an RS-232 connection that was converted to TCP/IP for consumption of the cybersecurity algorithms. This conversion process was performed by an FPGA running three softcore processors and preforming triple modular redundancy checking of the softcores to detect and isolate any compromised components.
    - Each of the cybersecurity algorithms used to protect against the outlined attack scenarios was composed of 500 lines of code or less to facilitate code validation.

A second prototype Sentinel was deployed on the ground. This highly simplified Sentinel was only used to provide security for the ground station, but related to the objectives of the flight evaluations, did not require any secure implementation features. This Sentinel included one component attached to the ground station used to transmit and receive information and commands from the UAV, and a second component to monitor the pilot's operator station. This component would monitor the ground station, as well as relay information from the Sentinel onboard the aircraft to the cybersecurity operator station capable of taking restorative action if a cyber-attack was detected.   This component attached to the pilot's station was used to monitor the pilot's commands to the aircraft.

The prototype Sentinel system included the following protections:

- To protect against cyber-attack scenario 1, the Sentinel onboard the aircraft would monitor the commands issued to the autopilot, as well as the state of the aircraft to determine if the commands were consistent with intended system operations. For example, under normal operations, a command to direct the onboard camera to begin tracking a target should not be immediately followed by a command to retract the camera system. In addition, the tracking by the onboard camera would normally be disabled before a command to retract the system is received. The Sentinel would defend against this cyber-attack by monitoring the commands issued to the payload and disabling commands that were determined to be inconsistent with the current state of the aircraft. In addition, if an inconsistent command was received, a cybersecurity operator on the ground was informed of the event.

- To prevent against cyber-attack scenario 2 the Sentinel onboard the aircraft was equipped with two GPS sensors. These sensors were used by the Sentinel to independently validate the latitude and longitude reported by the autopilot to the onboard camera system. The GPS hardware in the Sentinel was purposefully selected to contain a different hardware implementation than the autopilot's GPS implementation, so as to help avoid a supply chain injected attack. If the diversely redundant independent positional information deviated too far, the Sentinel could take actions to defend against the potential attack in real-time and support ground-based correction of corrupted metadata that would occur at a future date. To respond to the described attack, the Sentinel could replace the faulty positional data reported by the autopilot with data reported from a valid GPS device. A valid GPS device is a device approved by a majority vote (2 out of 3) of the available GPS devices; i.e., the autopilot's GPS and the two Sentinel GPS devices. To aid in correcting corrupted metadata at a future date the Sentinel records when a deviation is detected, the corrupted positional information, and the trusted positional information. All detections of a corrupted GPS were reported to a cybersecurity operator on the ground. In addition, the cybersecurity operator could, in real-time, alter the sensitivity of the algorithms used to validate the positional information.

- To protect against cyber-attack scenario 3, the Sentinel onboard the aircraft would coordinate with the Sentinel located on the ground to determine whether deviations from the current flight plan were initiated by a pilot. Whenever the pilot would initiate a change to the flight plan, the ground Sentinel would securely forward this information to the Sentinel onboard the aircraft. Any changes to the flight plan were reported to a cybersecurity operator station on the ground. If the alteration was due to a pilot's action, it would be reported as normal. If the alteration was due to a cyber-attack—i.e., no pilot action could be associated with the action—the cybersecurity operator was informed of the potential cyber-attack. At this point the cybersecurity operator could take a corrective action to restore the aircraft to the previous flight plan. For safety, the operator was limited to restoral actions that ensured that the aircraft stayed within the airfield and minimized the maneuvers needed to move between flight plans.

An important unanticipated outcome of the flight evaluation activity was the knowledge gained about assuring safety while conducting cyber-attacks during actual flights of UAVs at an active airport.

4.2 Development of the Serial Spy

The SerialSpy is a microcontroller-based device designed to monitor and/or manipulate RS-232 serial data between two devices. It uses a Microchip dsPIC33EP digital signal controller to passively monitor bidirectional serial data transparently from the host serial devices, insert new serial data packets, modify serial data packets, or completely block specific serial data. The SerialSpy is controlled through a USB serial interface or through telnet over Ethernet.

The SerialSpy is designed to be inserted between two devices connected via RS-232. In the UAV implementation of the Sentinel, the SerialSpy is inserted between the Piccolo autopilot and the camera gimbal as shown in Figure 44.

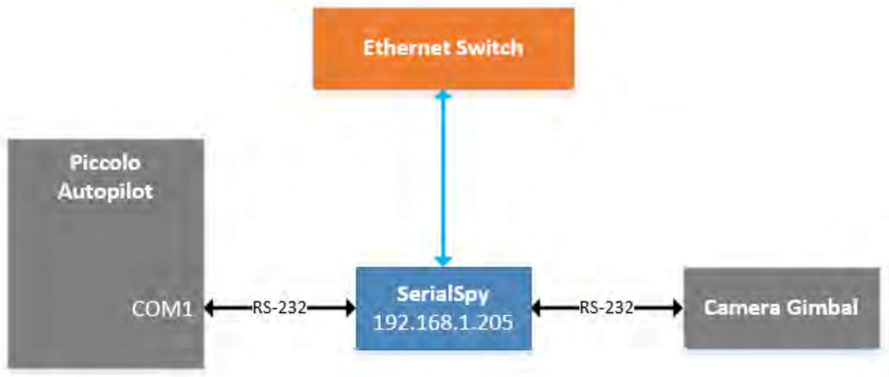


Figure 44. Simplified Block Diagram Showing SerialSpy Usage in a UAV Sentinel

The SerialSpy was specifically designed to overcome latency issues that arose with other methods of monitoring and modifying serial data. Previous design iterations of a serial communications monitor had used a Raspberry Pi single board computer and two USB to RS-232 adapters. A program written in C would monitor the serial ports and pass data between them passively in addition to modifying or blocking data as required. This approach had several disadvantages. First, the Raspberry Pi takes time to boot up before any such software could run, on the order of 30 seconds to 1 minute. This means that the autopilot and gimbal would be unable to communicate until the Raspberry Pi had fully booted and launched the software. Additionally, there is a certain latency and overhead associated with this approach. Each USB to serial converter adds several milliseconds of delay, and the Linux kernel also adds some delay. Although the processing time of the software is negligible, it must wait for the entire serial data packet to be received before re-transmitting the packet out the other RS-232 port.

The SerialSpy runs on a bare-metal microcontroller with no operating system, and so it boots up in milliseconds and immediately allows data to flow passively between the RS-232 ports. When not actively monitoring the data packets, the SerialSpy is able to pass data between the serial ports one byte at a time, effectively eliminating the delay. It does not have to wait and receive an entire serial packet before beginning to transmit data on the other serial port. For its RS-232 interfaces the SerialSpy uses DB9 connectors. P1 is a DB9-female connector wired as standard Data Terminal Equipment (DTE). P2 is a DB9-male connector wired as standard Data Circuit-terminating Equipment (DCE). The opposite gender connectors and appropriately crossed transmit/receive lines allow the SerialSpy to be inserted between two RS-232 devices. The SerialSpy can use baud rates from 1200 baud to 115200 baud.

Communications with the SerialSpy are provided by a USB-serial interface with a micro USB connector and an Ethernet interface with a standard RJ-45 connector. Both of these interfaces can be used to control and configure the SerialSpy. Commands include switching the device between passive and active mode, configuring the RS-232 baud rate, monitoring serial traffic in either direction, and configuring the various packet analysis modes for active mode.

As previously noted, the SerialSpy has two operational modes: passive and active. Passive mode will pass all data between the RS-232 ports without modifying or blocking any data. This mode is protocol agnostic; any format of data can pass through and be monitored. Active mode enables the SerialSpy's data processing module, which can modify or block serial data. Active mode is protocol-dependent and requires specific code to be written for protocol analysis. In either mode, the data can be monitored through the USB port or over Ethernet.

For the UAV implementation of the Sentinel, the data passing between the autopilot and the camera gimbal used the Piccolo gimbal communications protocol[3]. This protocol defines the specific packet structure of the data that includes a start-sequence header and length byte for each packet. The SerialSpy can be configured to be strict and only allow valid packets to flow through, or to allow all packets through, even if they are malformed.

For defense against cyber-attacks the SerialSpy was configured to search for specific packet types. Table 1 lists the packet types that were analyzed.

Table 12. Commands that the SerialSpy searched for in Gimbal Communications

Group	Cmd	Description	Actions
0x10	0x47	Gimbal telemetry	Compare GPS coordinates to truth data, alter if needed
0x10	0x10	Host GPS Data	Compare GPS coordinates to truth data, alter if needed
0x10	0x40	SPOI command	Block or alert when detected
0x00	0x45	SPOI command	Block or alert when detected
0x00	0x46	SPOI command	Block or alert when detected
0x00	0x43	Gimbal retract	Block or alert when detected

Messages that contained GPS coordinates from the autopilot were compared to known "truth data", which was provided over Ethernet from the SiCore computer portion of the onboard Sentinel. If the GPS coordinates were detected to be greater than 100 meters from the truth data, an alert was sent to the

[3] Vaglienti, B. (2011). *Gimbal Communications v2.2.0.d.* Cloud Cap Technology

Sentinel. Optionally, the correct GPS coordinates provided by the SiCore computer could also be inserted into the datastream for the camera gimbal to use as metadata.

To defend against SPOI attacks and gimbal retract attacks, the operator could elect to have commands for SPOI and gimbal retract either completely blocked from being transmitted or to have an alert sent to the Sentinel over Ethernet

4.3 Development of the Tester's Interface

The tester's interface (shown in Figure 45 as the Test Director Station) was developed primarily to allow the test director to monitor the aircraft's true state while it is undergoing a cyber-attack. For example, the waypoint attack takes command of the UAV's flight plan while masking the attack on the operator's ground control station. As a result of the masking, the operator's display shows the aircraft on the intended route while, in reality, the aircraft's flight path is being rerouted. The tester's interface takes advantage of GAUSS' dual radio links for command and control operating at 450 and 900 MHz. Two separate instances of the operator interface software are run at the ground control station, one for each frequency. While the masking attack is underway on the 900 MHz ground control station, the tester's interface uses the 450 MHz control station to monitor the true status of the aircraft.

The situational display is based on GTRI's FalconView map display software. A FalconView plugin visualizes the current state of the system (see Figure 45). As indicated by the legend in the upper left corner of the display, the interface can show where the aircraft really is (Truth, in blue), where the aircraft operator sees it on his Piccolo Command Center (PCC) and where camera operator sees it on ViewPoint while the attack is happening (Attack, in red), and where the defensive systems indicate the aircraft's correct location is (Defense, in green).

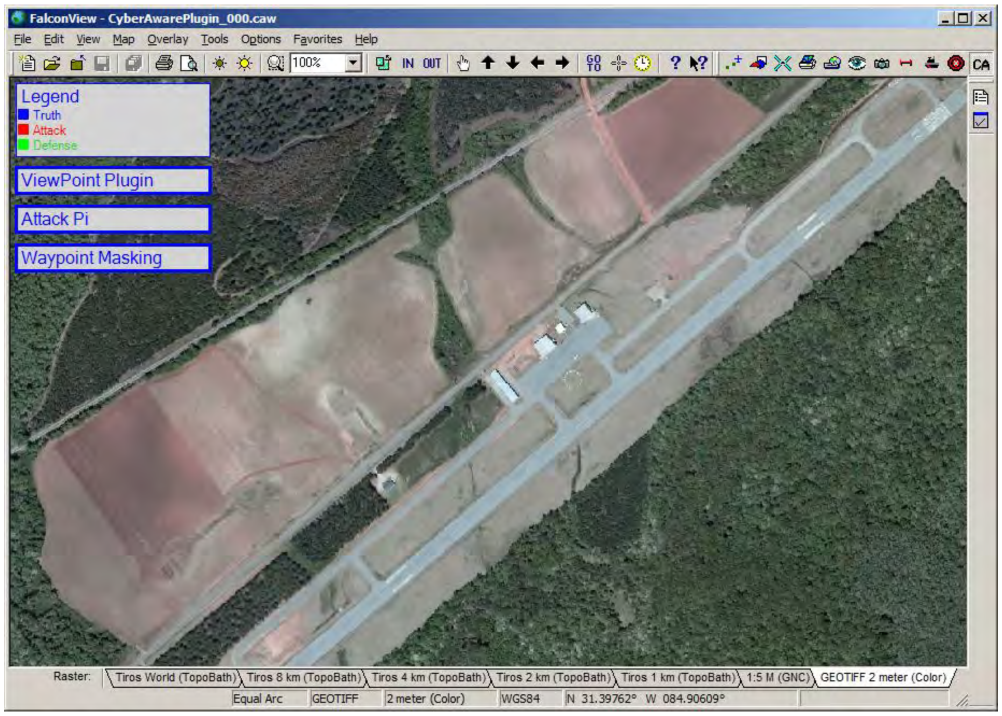

FalconView - CyberAwarePlugin_000.caw
File Edit View Map Overlay Tools Options Favorites Help
100%
IN OUT
CA
Legend
🟦 Truth
🟥 Attack
🟩 Defense
ViewPoint Plugin
Attack Pi
Waypoint Masking
Raster: Tiros World (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) GEOTIFF 2 meter (Color)
Equal Arc GEOTIFF 2 meter (Color) WGS84 N 31.39762° W 084.90609°

Figure 45. FalconView Display on the Tester's Interface

The HUD indicators below the legend give information about the status of various applications. When the FalconView plugin is actively communicating with those components, the HUD boxes expand as shown in Figure 46. The Component Indicators light up for each component when an attack or defense occurs. These components, ViewPoint Plugin, Attack Pi, and Waypoint Masking are described in detail later in this section.

Figure 47 shows the Tester's Interface displaying the Aircraft Indicators. These are aircraft icons in different colors that indicate the aircraft's true position, attack position, and defense position. The aircraft's true position shown in blue indicates where the aircraft really is. This data is retrieved from the 450 MHz Piccolo ground station. The attack position, shown in red, indicates where the PCC is displaying the aircraft's position during a waypoint attack with masking. It also indicates where ViewPoint is showing the aircraft position in the case of the GPS walk-off attack. The defense position, shown in green in Figure 48, indicates the aircraft location as determined by the Sentinel in response to a waypoint or GPS walk-off attack.

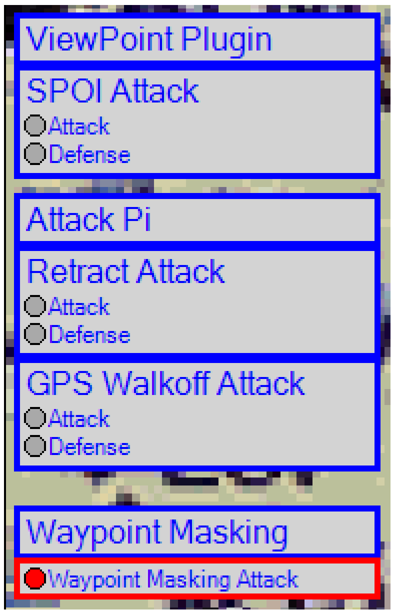


Figure 46. HUD Indicators

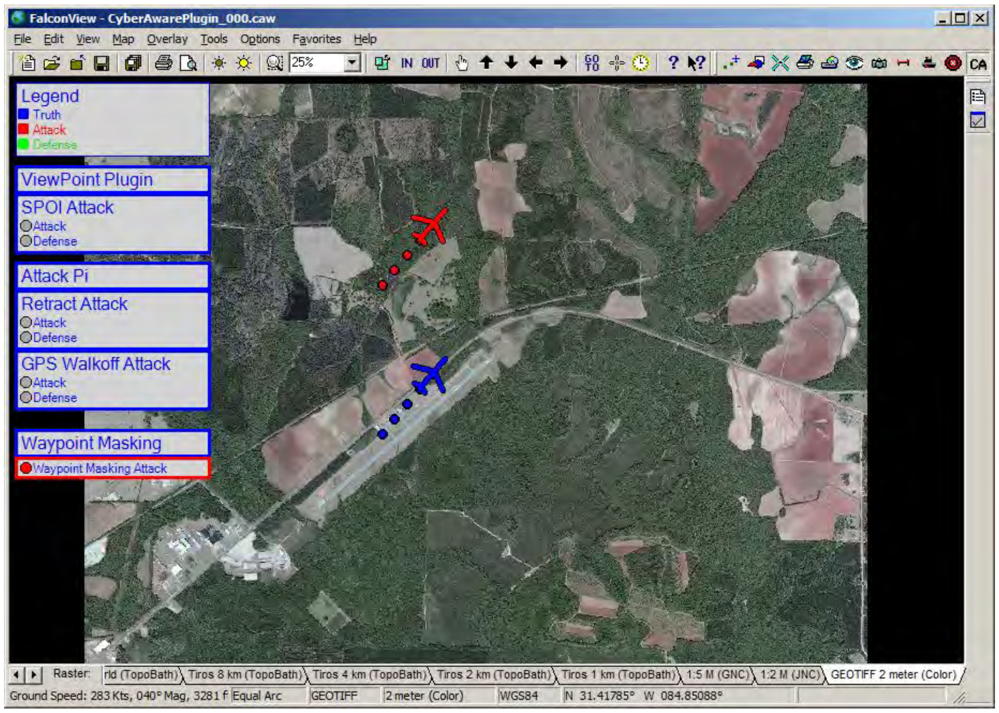

FalconView - CyberAwarePlugin_000.caw
File Edit View Map Overlay Tools Options Favorites Help
Legend
Truth
Attack
Defense
ViewPoint Plugin
SPOI Attack
Attack
Defense
Attack Pi
Retract Attack
Attack
Defense
GPS Walkoff Attack
Attack
Defense
Waypoint Masking
Waypoint Masking Attack
Raster: rld (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) 1:2 M (JNC) GEOTIFF 2 meter (Color)
Ground Speed: 283 Kts, 040° Mag, 3281 f Equal Arc GEOTIFF 2 meter (Color) WGS84 N 31.41785° W 084.85088°

Figure 47. Tester's Interface Showing Truth and Attack Positions for the Aircraft

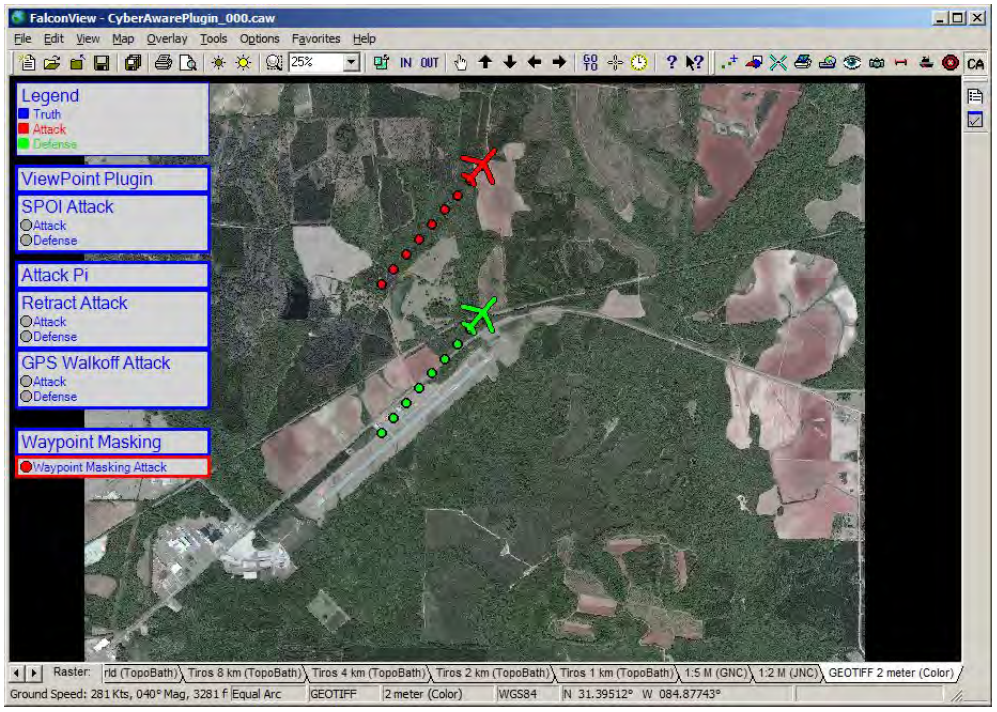
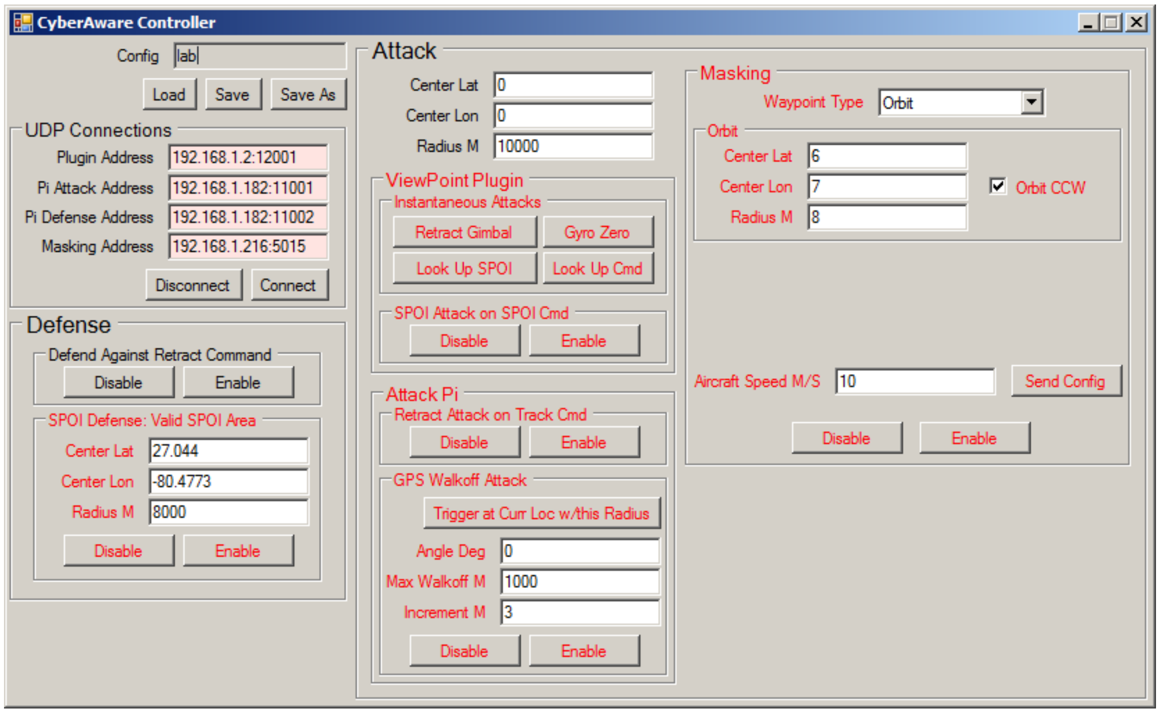

FalconView - CyberAwarePlugin_000.caw
File Edit View Map Overlay Tools Options Favorites Help
Legend
Truth
Attack
Defense
ViewPoint Plugin
SPOI Attack
Attack
Defense
Attack Pi
Retract Attack
Attack
Defense
GPS Walkoff Attack
Attack
Defense
Waypoint Masking
Waypoint Masking Attack
Raster: rld (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) 1:2 M (JNC) GEOTIFF 2 meter (Color)
Ground Speed: 281 Kts, 040° Mag, 3281 f Equal Arc GEOTIFF 2 meter (Color) WGS84 N 31.39512° W 084.87743°

Figure 48. Tester's Interface Showing the Defense Position for the Aircraft (in green)

The tester's interface can also be used to set up several of the cyber-attack scenarios. The main menu for setting the various parameters for an attack is shown in Figure 49. Starting in the upper left corner of the menu, the section labeled "UDP Connections" (see Figure 50) controls the UDP connections to various components in the system. In particular, the "Plugin" refers to the ViewPoint plugin software used to implement the as described in Section 3.2.3.4. The "Pi Attack" refers to the software on Attack Pi 2 used for the GPS walk-off attack. The "Pi Defense" refers to the software on Attack Pi 2 that could be used to defend against the GPS walk-off attack. However, the defense against the GPS walk-off was also available on the SerialSpy (at address 192.168.1.205) and during testing it was decided that the attack software and the defense software should be hosted on separate devices. This was done to avoid potential confusion over the roles of the various components for attack and defense. The "Masking Address" refers to the Masking Attack Pi at the ground station as shown in Figure 6.

When the UDP receive and transmit sockets are successfully created, the text box turns from red to green as shown in Figure 50 (b).


CyberAware Controller

Config: lab

UDP Connections
- Plugin Address: 192.168.1.2:12001
- Pi Attack Address: 192.168.1.182:11001
- Pi Defense Address: 192.168.1.182:11002
- Masking Address: 192.168.1.216:5015

Defense
- SPOI Defense: Valid SPOI Area
  - Center Lat: 27.044
  - Center Lon: -80.4773
  - Radius M: 8000

Attack
- Center Lat: 0
- Center Lon: 0
- Radius M: 10000
- Attack Pi / GPS Walkoff Attack
  - Angle Deg: 0
  - Max Walkoff M: 1000
  - Increment M: 3

Masking
- Waypoint Type: Orbit
- Orbit
  - Center Lat: 6
  - Center Lon: 7
  - Radius M: 8
  - Orbit CCW: ✓
- Aircraft Speed M/S: 10

Figure 49. Tester's Interface

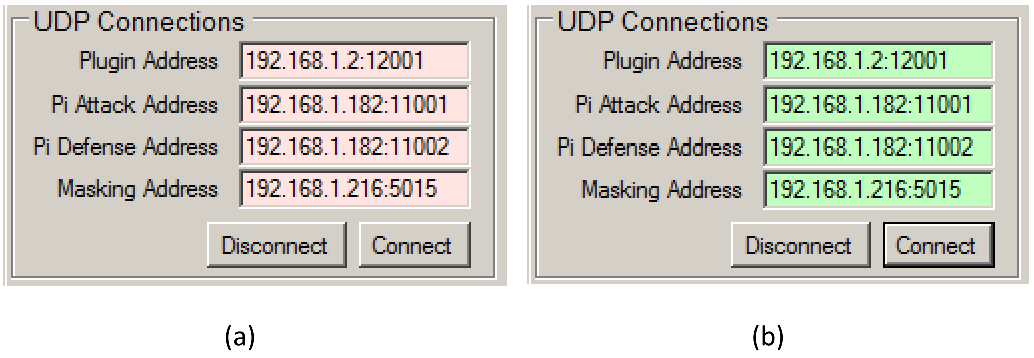
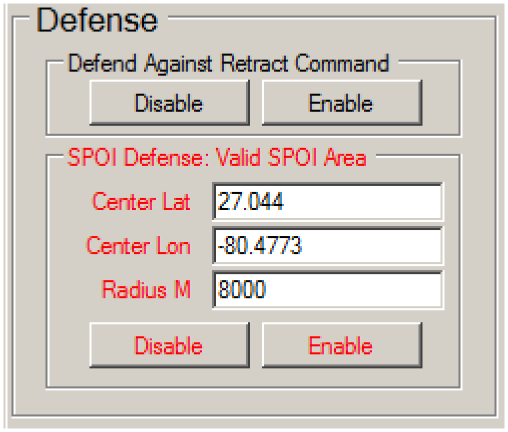

UDP Connections
- Plugin Address: 192.168.1.2:12001
- Pi Attack Address: 192.168.1.182:11001
- Pi Defense Address: 192.168.1.182:11002
- Masking Address: 192.168.1.216:5015

Figure 50. UDP Connections Control (a) Sockets not Connected, (b) Sockets Connected

The Defense section (Figure 51) controls the Defense Software component located on the Attack Pi.

i. Defend Against Retract Command - This defense removes all retract commands from the command stream as it heads to the gimbal. This defense can be enabled or disabled.

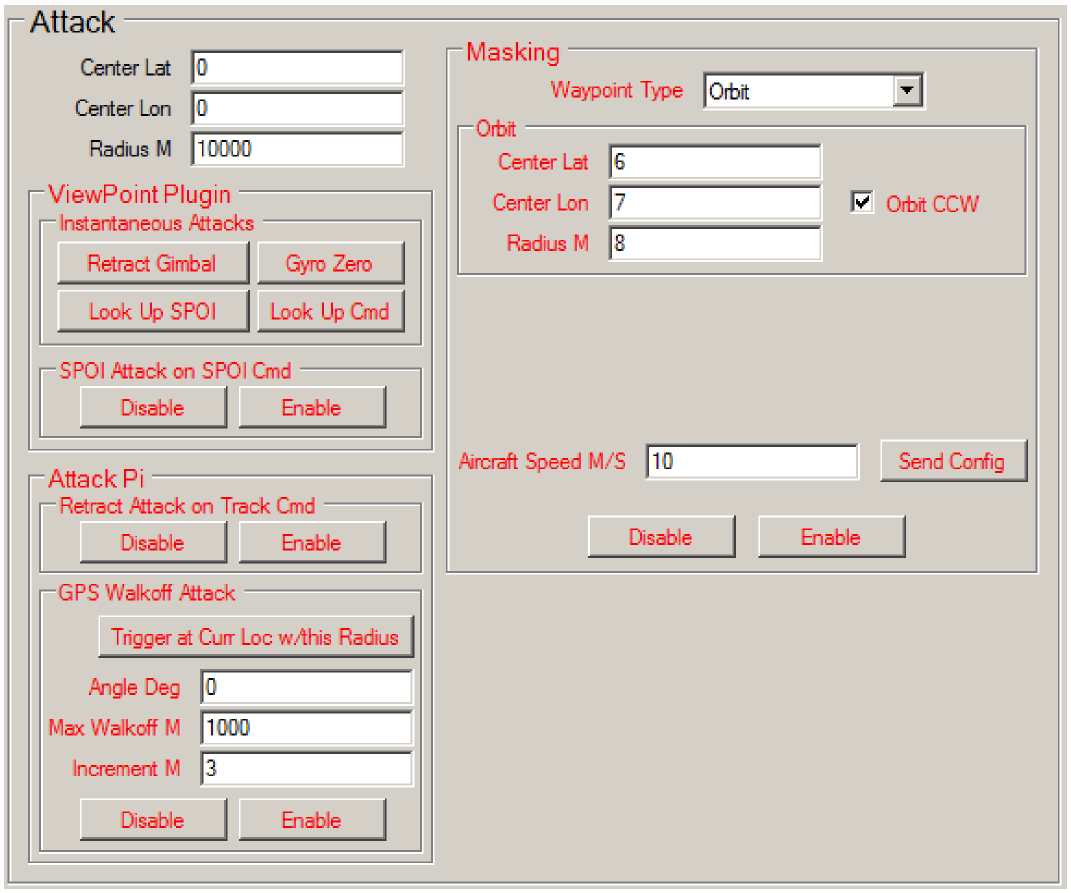
ii. SPOI Defense: Valid SPOI Area - SPOI commands should only place the camera within the mission area. This section allows the user to specify the mission area where SPOI commands are valid. When enabled, this defense will remove all SPOI commands with an altitude above the current aircraft altitude or outside the mission area.



Center Lat 27.044
Center Lon -80.4773
Radius M 8000

Figure 51. Defense Section of the Tester's Interface

The Attack section controls with the ViewPoint plugin attack component (gimbal command attacks) and the Attack Pi 2 attack software (GPS walk-off). The "Center Lat" and "Center Lon" parameters allow the tester to set the latitude and longitude of the center of the geographic area in which attacks will occur. The "Radius" is used to specify the radius of the attack region.



Attack

Center Lat: 0
Center Lon: 0
Radius M: 10000

ViewPoint Plugin
- Instantaneous Attacks
- SPOI Attack on SPOI Cmd

Attack Pi
- Retract Attack on Track Cmd
- GPS Walkoff Attack
  - Angle Deg: 0
  - Max Walkoff M: 1000
  - Increment M: 3

Masking
- Waypoint Type: Orbit
- Orbit
  - Center Lat: 6
  - Center Lon: 7
  - Radius M: 8
  - [✓] Orbit CCW
- Aircraft Speed M/S: 10

Figure 52. Attack Section of the Tester's Interface

The section labeled "ViewPoint Plugin" (see Figure 53) is used to control the gimbal command attacks. By clicking the appropriate button, the tester can initiate several instantaneous attacks on various gimbal commands:

> Retract Gimbal – sends a retract gimbal command which causes the gimbal to retract into the aircraft's fuselage
>
> Gyro Zero – sends a Gyro Zero command zeros all the gimbal angles and throws off the calibration of the gimbal
>
> Look Up SPOI – send a command to point the camera directly upward using the sensor point of interest (SPOI) gimbal command message
>
> Look Up Cmd – sends a command to point the camera directly upward using the camera angle command message

The SPOI Attack on SPOI Cmd section allows the tester to enable or disable the SPOI attack. Enabling this attack causes the camera to point upward whenever the payload operator tries to track a spot on the ground using the SPOI command. The attack modifies the SPOI command by setting the point of interest directly above the aircraft's location so that it will point upward instead of pointing at the correct location on the ground.

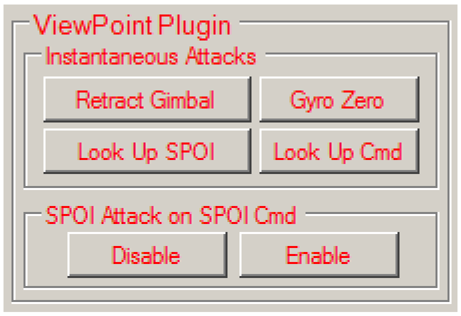
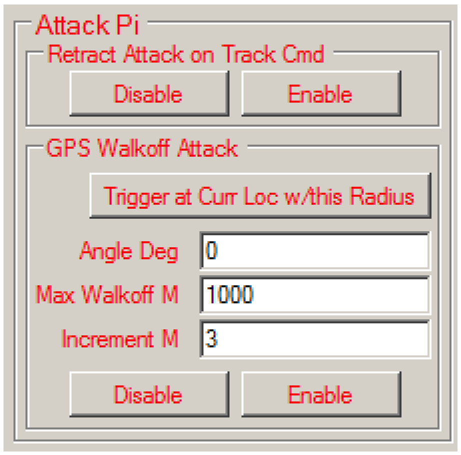


Figure 53. ViewPoint Plugin Section

The "Attack Pi" subsection (Figure 54) is used enable a gimbal retract attack and to set up and enable the GPS walk-off attack. The "Retract Attack on Track Cmd" section allows the tester to enable or disable the gimbal retract attack. When the attack is enabled and the aircraft enters the geographic region for the attack, the camera gimbal retracts whenever the operator tries to engage the tracking feature. The ViewPoint plugin with the attack software (hosted on the Test Director station) sends a retract command to the gimbal when the operator clicks on a spot in the video in order to lock the camera on an object for tracking. As a result this commonly-used gimbal feature is rendered useless unless the corresponding defense is active on the SerialSpy.

The "GPS Walkoff Attack" subsection controls how the GPS position that is being fed from the autopilot to the camera gimbal is corrupted. The GPS walk-off attack offsets the true GPS location of the aircraft by the specified Increment in the compass direction specified by Angle every time a GPS update occurs. The "Max Walkoff" limits the magnitude of the position error. The corrupted data is then passed on to the camera gimbal and is included in the video metadata that is sent to ViewPoint. Recall that the location and radius in which this attack will be triggered is set at the top of the Attack section (see Figure 52).

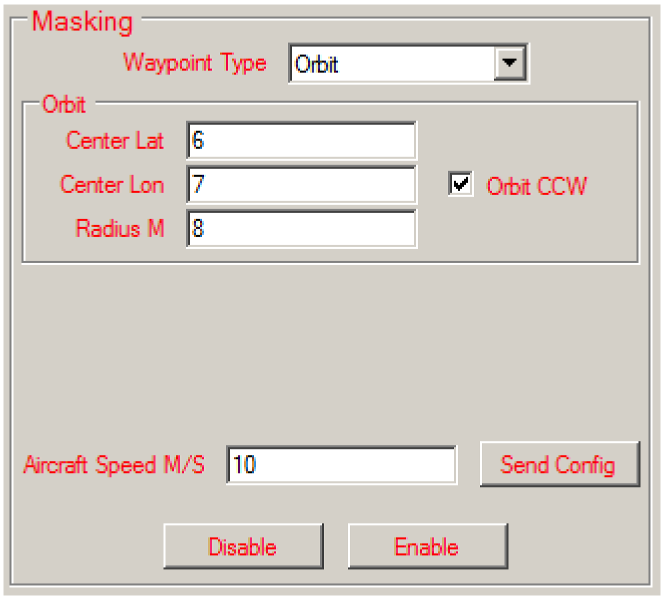
The "Trigger at Curr Loc w/this Radius" allows the tester to initiate the GPS walk-off attack at the aircraft's current location rather than waiting until the aircraft enters the attack region. It does so by setting the center of the attack region to the aircraft's current location.


Attack Pi
Retract Attack on Track Cmd
Disable
Enable
GPS Walkoff Attack
Trigger at Curr Loc w/this Radius
Disable
Enable

Angle Deg: 0
Max Walkoff M: 1000
Increment M: 3

Figure 54. Attack Pi Section

The "Masking" section of the Tester's Interface, shown in Figure 55, allows the tester to set up and enable the waypoint attack with masking. This attack hijacks the aircraft while hiding the attack from the aircraft operator on the Piccolo Command Center.



Waypoint Type: Orbit
Center Lat: 6
Center Lon: 7
Radius M: 8
[✓] Orbit CCW
Aircraft Speed M/S: 10

Figure 55. Masking Section of Tester's Interface (Orbit Waypoint Selection)

The "Waypoint Type" drop-down menu lets the tester to specify whether the waypoints for hijacking the route are either an orbit waypoint or a waypoint list (for a route). If a orbit waypoint specified, the tester can set the latitude and longitude of the center of the orbit and the radius of the orbit. If the tester wants to send a list of false waypoints, he can create a list by adding waypoints and specifying their latitude and longitude (Figure 56).

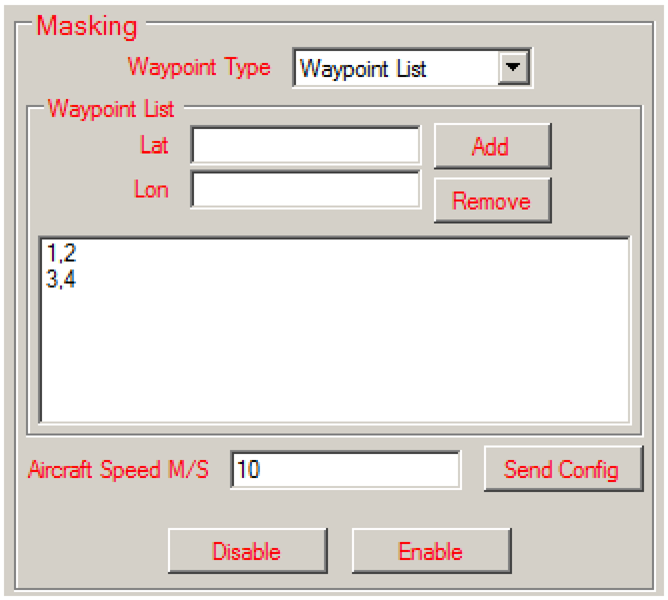


Figure 56. Masking Section of Tester's Interface (Waypoint List Selection)

The "Aircraft Speed" box allows the tester to set the speed of the aircraft once the attack has taken control of the aircraft.

4.4 Aircraft Integration

Integration of the Sentinel components in GTRI's GAUSS air vehicle required the design and fabrication of wiring harnesses, additional GPS antenna mounts, and a chassis to hold all of the components (Figure 57). In addition, a power kill switch was added to the system to enable the flight test engineer to kill power to all of the payload components via a discrete command from the autopilot system. This was done to ensure that all of the non-flight critical components communicating with the autopilot could be killed in the event of an emergency.

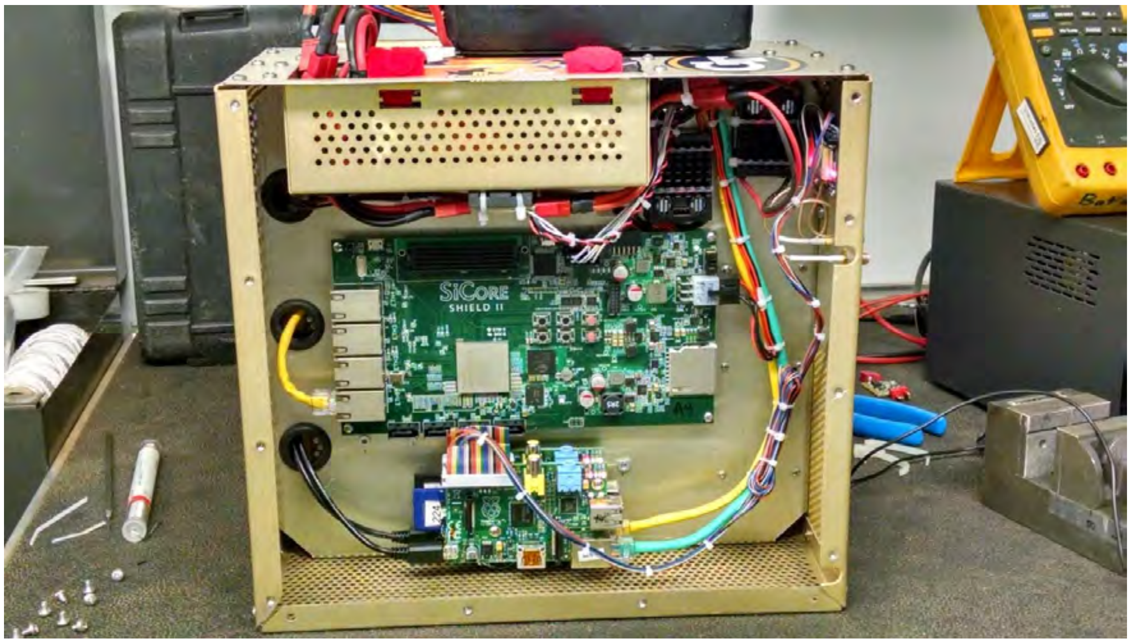


Figure 57. Payload Chassis

4.5 Flight Test Demonstration Attacks

The flight test demonstration was conducted during the week of Oct. 6 – 10 at the Early County airport in Blakely, GA. Three types of attacks were successfully demonstrated: a GPS walk-off, a gimbal command attack, and a waypoint attack.

4.6 Sentinel Protections For Selected Attacks

4.6.1 Gimbal Attack Detection and Mitigation

Degradation and denial of service attacks are possible because the gimbal trusts the sender of any commands that it receives. To prevent this type of attack, the system should be able to evaluate any command it receives to determine its validity.

Changes cannot be implemented in how the ViewPoint software issues commands or how the gimbal responds to them because they are commercial products and the source code is not available. However, it is possible to place a piece of in-line hardware or software on the UAV that receives the command packet before the gimbal and can decide whether or not to forward it along to the gimbal based on mission conditions.

Several methods can be used to make decisions on the validity of gimbal commands. One method to help catch unauthorized commands is to implement an authentication scheme, possibly by appending a cryptographic signature to messages sent from the ViewPoint software to the gimbal. However, this will not protect the gimbal system from the case in which a malicious agent has compromised the Piccolo

Command Center (PCC). Depending on the degree of compromise, the malicious agent could still be able to send messages that the UAV would consider authentic.

In a similar manner, providing additional authentication to commands capable of causing damaging effects would be helpful but not sufficient. An attacker who has not fully compromised PCC to the point of recovering the cryptographic key would be halted by such a defense, but further compromises may render this ineffective.

To protect from compromises of PCC the UAV should be able to judge the legitimacy of commands. To do this a run-time analysis can be performed to determine whether or not executing a command makes logical sense based on mission context. For example, if the system received a command to retract the gimbal while it is in a pre-specified area of interest, an intelligent decision would be to not immediately trust the command and attempt to verify its authenticity. In addition, authorized operators should be able to issue whatever commands they need, so there must be an override capability to verify that commands that may seem illogical are in fact legitimate.

4.6.1.1 Using Mission Context to Detect Gimbal Attacks

Cloud Cap software provides flexibility for a wide variety of mission operations, which makes the system susceptible to insider attacks involving seemingly valid commands that interfere with user operations. To prevent these, systematic rules based on mission context have been developed to limit when and where certain commands should be considered authentic.

The following algorithms use structures and methods from the software development kit provided by Cloud Cap for ViewPoint plugin creation and are aimed toward detecting the attacks that have been found to be the most feasible. Despite the following algorithm being written using an SDK, one could decode the information bytewise from the message streams and follow the same algorithms.

4.6.1.2 Packet Detection

The method LookForGimbalPacketInQueue() searches through a queue of packets and determines if a packet of gimbal type (i.e., a gimbal packet) is present in the message stream. It then stores this packet in a predefined buffer. The packet is then inspected to see if the packet type is a gimbal command. Almost all of the gimbal command vulnerabilities fall into this type with the exception of the user-warning packet.

4.6.1.3 Retract/Deploy Command Detection

The gimbal packets are further inspected to determine if the packet group is that of *Gimbal command and control group*. If so, then it is passed to the method that checks if it can be decoded into a retract/deploy struct pointer. If the method returns false the packet is ignored and the monitoring of the stream for packets continues. If the method returns true then the stream is decoded into information determining whether the gimbal is being commanded to either retract or deploy.

Under the assumptions that normal operations would entail the retraction and deployment of the gimbal directly after take-off and directly before landing, the velocity of the gimbal relative to Earth and the distance of the gimbal from the ground station should be relevant criteria to determine whether the gimbal retract/deploy command appears to be authentic.

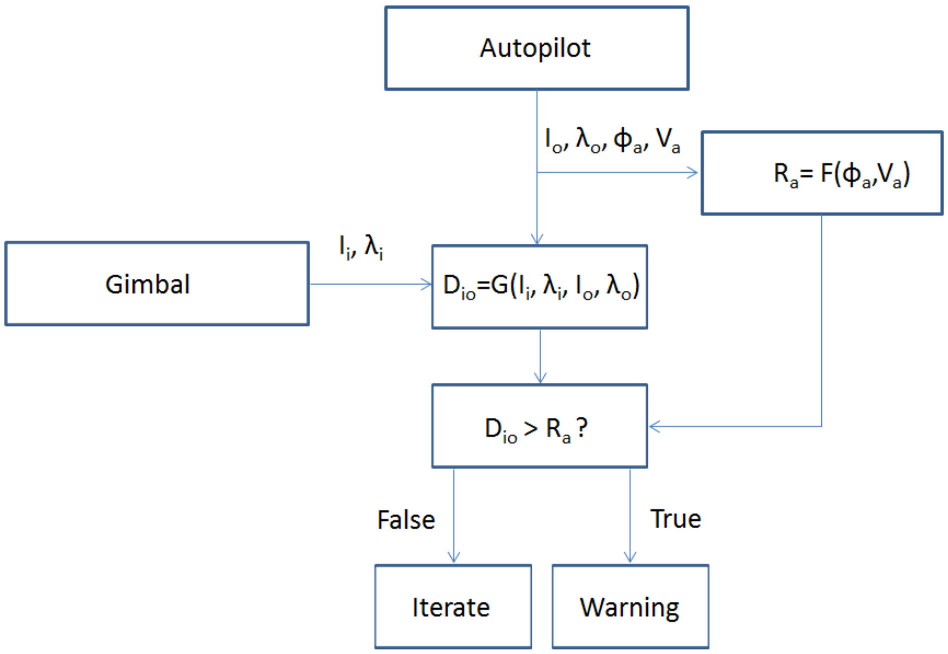
The aircraft velocity and position can be determined by monitoring the gimbal telemetry stream for packets of type HOST_GPS_DATA_GIMBAL_PKTTYPE and of group GIMBAL_POSITION_INFORMATION_GROUP. These telemetry packets can be decoded to give the GPS position and velocity of the aircraft. These two pieces of information can be used to determine what phase of flight the aircraft is in. If the phase is take-off or approach/landing, then the retract/deploy command is considered authentic. If the aircraft is in cruise or loiter mode then the retract/deploy command should be considered malicious.

4.6.1.4 Erratic Gimbal Command Detection

To protect against a Gimbal Command attack it is assumed that during normal operations the gimbal should never be slewed to view a location above the horizon. Similar to the process previously described, the telemetry stream is checked for gimbal packets in the queue. The method DecodeGimbalCmdPacket() is used to give an elevation angle of the gimbal. The GPS information is used to determine the aircraft altitude. If the aircraft altitude is higher than the altitude at which the gimbal is pointed, then the command is authentic. If the gimbal is pointed at a higher altitude than the aircraft then the command is considered malicious and the user can be warned via a message sent through a payload message stream using the autopilot command and control link.

Further constraints can be placed on the gimbal angles by limiting the gimbal orientation based on mission CONOPS. For example, if the UAV mission is to loiter overhead a specified target then the gimbal field of view should never extend outside the orbit of the aircraft.

A simple diagram illustrates the application of mission context to limit the functionality of the gimbal. With this limited functionality, if the gimbal deviates from its intended use then a warning message will be emitted to the Cyber Commander. Using the image centroid location and intercepting the vehicle center of orbit position, a comparison will be done to ensure that the tracked object is within the orbit's radius. This makes sense if the mission context is to circle overhead an object for recon purposes.



l_i, λ_i : latitude, longitude of image
l_o, λ_o: latitude, longitude of orbit center
R_a : aircraft orbit radius
D_{io}: distance of image to orbit center
V_a : aircraft velocity
ϕ_a : aircraft bank angle

In steady, level flight the relationship between the aircraft's bank angle ($\square_a$), velocity ($_{Va}$), and turn radius (R_a) can be expressed as

$$R_a = \frac{V_a^2}{g * \tan(\phi_a)}$$

where g is the gravity force.

The function defined as G relates the latitude and longitude of the gimbal and orbit centroid to a distance. This is done using an ellipsoidal Earth (WGS-84) model and calculating the distances using Vincenty's formulae.

4.7 Cyber Commander Interface Design and Development

During Phase 1, the Cyber Commander interface lacked flexibility and displayed attack and restoration information in a manner confusing to non-developers. Additionally, only one aircraft could be plotted on the map. Increasing the number of planes involved manually encoding a unique aircraft into the test telemetry data. Initial development efforts focused primarily on increasing the flexibility of the existing interface rather than adding new UI elements and features.

Recognizing the need for monitoring multiple aircraft simultaneously, we included an aircraft table in the database structure. This table stored both a 32-digit hexadecimal UUID in canonical form. We added URL commands for both inserting a new aircraft and deleting existing ones. AJAX calls are used to check for the addition of new aircraft. If one is found and corresponding telemetry data is available, then plane symbol is plotted on the map to represent the aircraft's movement. Movement is defined as a redrawing of the aircraft icon using new telemetry data read from the database. The initial design moved chronologically through the database, grabbing one row of telemetry information every 2 seconds. In our initial modifications to the code, we read in one row at a time, but used the difference in timestamp between the current row and the next row in the telemetry table to dynamically adjust the timing of the AJAX function responsible for moving the aircraft.

A crucial capability of the cyber commander is to verify this aircraft movement against an existing flight plan, consisting of a closed loop of waypoints connected with directional arrows. We added a table containing waypoint information to the database structure, as well as a URL command for saving new waypoints or editing existing ones. This waypoint data is used for three primary purposes in the front-end display. First, the latitude and longitude attributes allows for Google map markers to be plotted. Second, the specified destination value in the ingested telemetry data is used to draw a path to an aircraft's current destination, mirroring functionality included in the Piccolo Command Center. Third, the combination of destination, latitude, and longitude data is used to draw directional arrows between waypoints on the Google map display.

An important requirement of the cyber commander interface is its ability to update in real time. We elected to use recursive AJAX calls to repeatedly update features on the map at set intervals. A side effect of this implementation was that the number of map features gradually increased over time, leaking memory and crashing the browser. To remedy this problem, we decided to completely delete and redraw all elements each time the AJAX functions refreshed. For the directional paths connecting the waypoints, this logic worked sufficiently. Infrequently, however, an arrow is not drawn, an issue we likely attribute to the single-threaded nature of JavaScript. Waypoints had to be handled in a different manner. Complete redrawing of waypoint markers introduced a highly distracting 'flickering' effect on each refresh. To address this issue, we introduced logic that cross checks the waypoints stored in the interface against those returned from the database. If any differences are identified, then only the changed waypoints are altered. In future iterations of the GUI, we would like to apply the same cross checking logic to the directional arrows connecting the waypoints.

4.7.1 Development of Interface

Once the original code base had been modified to handle an arbitrary number of aircraft and flight plan markers, we focused our efforts on developing a more user-friendly interface for the cyber commander station. Our first brainstorming session produced an interface quite similar to the current GUI implementation. We used the Balsamiq mockups tool to create and refine a representation of the interface.

We drew heavily on the usability heuristics as described by Jakob Nielsen in crafting the original design.

4.7.2 Visibility of system status

The left hand terminal existed in the original mockup and is meant to keep the user informed of the system status at all times by signaling both expected and unexpected aircraft commands.

4.7.3 Match between system and the real world

Additionally, the displaying of aircrafts and navigation of the Google map are consistent with applications used for monitoring geographic areas. It is natural for the cyber commander to assume a 'bird's eye view' of the monitored aircraft. Throughout the interface, terminology was selected to match the vocabulary associated with the domain of aircraft surveillance.

4.7.4 Flexibility and efficiency of use

We addressed system flexibility by providing multiple points of entry into the alerts stored in the system. For example, a user can select an alert from the left terminal or open a search window to manually filter for specific types of alerts. This filtering uses the aircraft id value, the timestamp of the alert, and the alert type. To accommodate more advanced users, we allow direct access to an aircraft's alerts by clicking on the corresponding map icon.

4.7.5 Aesthetic and minimalist design

Information visible to the user is kept as minimal as possible. We strictly defined the types of messages that can be displayed, which helped preserved this minimalism. At the highest level, the user can see aircraft names, positions, and movement, as well as a recent history of command and attack alerts; this information is consistent with the monitoring goals of the cyber commander.

4.7.6 Recognition rather than recall

Recognizing that distinguishing between these hexadecimal numbers put a large strain on an operator's working-memory, we decided to incorporate a more understandable front end identifier. For example, instead of a 32-digit hexadecimal identifier, the aircraft would be recognized as "aircraft1," or any other arbitrary name. This means the possibility of duplicate names exists, but the operator can still distinguish aircraft by clicking the corresponding map icon and seeing their unique ID values.

Difficulties arising during development stemmed from web browser idiosyncrasies. The current iteration of the interface functions in both Firefox and Google Chrome web browsers. Initially, we experienced some difficulty in transitioning from Firefox to Chrome; the browsers have different protocols for handling user selection from drop down lists. Memory management in both browsers is also handled by in fundamentally different way. At one point during development, a small error in the code resulted in a difficult to find recursive memory leak. In chrome, the Windows task manager showed a slight, almost imperceptible increase in memory usage over time. Firefox, however, suffered immediate performance decreases as the allotted memory rapidly increased in size. Once we resolved these issues, the interface functioned equivalently on both browsers.

Future work involves fully fleshing out planned features that depend on the communication protocol with the monitored aircraft, and the type of hardware information that a user would want to access. For example, a 'Home' and 'Abort' button are included in the interface's maps. Ideally, in an emergency situation, an operator could either halt flight commands or force an aircraft to return to its deployment site. Understandably, this feature would require deep knowledge of the protocols used for communicating remotely with the aircraft. The backend database affords the storage of a variety of relevant hardware information. For example, the database could be expanded to store more detailed aircraft information such as physical dimensions or engine type. Elements have been added to the interface to allow a user to query for this information, but with an empty database these elements are currently nonfunctional.

The Django backend handling the routing of incoming URL commands currently runs using the provided development server. Before final implementation, a shift to a more robust and secure production server such as Apache 2 is essential. From testing perspective, the lightweight Django server functioned well in our highly dynamic development environment. Django's method of serving static files to clients caused frustration in initial attempts at transitioning to a production environment.

Future iterations of the interface would benefit greatly from usability testing. The current interface has only been handled directly by the development team, so testing would help ensure that the system image provides the necessary information for users to form a conceptual model consistent with that of the team. Feedback from the usability studies will be essential to finely tuning the target users' interactions with the system.

For the sake of simplicity, we constrained the interface development to work on a laptop or desktop web browser environment. We can see tremendous value in expanding our design into the mobile market, allowing the cyber commander to monitor aircraft status on a tablet device such as an Apple iPad. We imagine the first iteration of this expansion being an application that can be accessed through the tablet device's browser. Future work could also include the development of native applications that could better leverage the available hardware resources on mobile devices. Again, these new interfaces would require usability tests and modifications, as a touch input method offers new challenges not apparent when using a touchpad or mouse.

4.7.7 RabbitMQ

A big part of the project was using the RabbitMQ messaging service which served as messaging system between the interface and the Sentinel. Essentially, the RabbitMQ delivers messages to locations same binding key as the routing key from which the message was sent. A producer program from a part of the sentinel (e.g. Gps Algorithm, Waypoint Algorithm, etc.) would send comma-separated messages to front end where a consumer program would read them, interpret them and call a URL with the information to store a database eventually.

Upon start of the consumer program, a producer code block requests a list of waypoints that will be sent immediately back by another part of the system. A problem that occurs is that the aircraft will not be shown on the screen without a set of waypoints. Additionally, if there is an error in the sequence of waypoints (i.e. a waypoint is missing or the waypoints are not in the logical sequence), the aircraft will again not populate on the screen. The interface should be modified to show the aircraft independently of whether a correct set of waypoints are in the database. Further investigation is needed to accomplish such a task, but on first thought, it might be possible to override the requirement of the system to display waypoints. What is happening is there is logic flaw in the javascript code which creates an error to the whole website. To possibly override such an error, a series of if-statements could fix the interface to display waypoints when only correct ones are stored. Additionally, it might be helpful to have other parts of the system to check for correct waypoints to better insure the reliability of the interface.

The consumer program has a set of binding keys that are constantly being read to check if messages have arrived from the various sources in the sentinel. A message from the same binding key will have distinct information that is separated by commas. This allows for easy parsing of the message into individual pieces of data. From there, the program will concatenate the data into a suitable URL and temporarily opens the URL to transport the information into the database. The process of URL calls is the convenience of Django. Of course, messages from different binding keys will contain different data which eventually call different URL. Therefore, the keys need to be unique as they are now. Problems can also occur if there is not the same binding key in the consumer as the routing key in the producer. Challenges arose when spelling errors occurred, or specific protocols were not established. A major issue that happened was when the data is not in the specified format. Examples include data in the wrong order, missing data, or the data was not compatible with the URL regular expressions that interpreted the URL calls. For the future, a more robust way of interpreting the data needs to be used. Unfortunately, it is inherent to the RabbitMQ to have exactly the same binding and routing keys.

General Comments: The RabbitMQ is a central part of the system because it delivers essential messages from each part eventually ending up in the front end interface. It is uncertain whether the current security on the RabbitMQ is secure enough. If the RabbitMQ was attacked, that would compromise significantly what the operator sees. A recommendation is to run additional testing on how the RabbitMQ could be compromised. If the results are significant enough to be concerned, additional security should be added or a new way to send messages should be implemented. Security methods for the messaging needs to be researched further.

4.7.8 GPS Parameter Display

On a separate display away from the Cyber Command display, an operator can enter values for the alpha and sliding window that the GPS algorithm incorporates when determining if there is an attack. As a result, the Cyber Command operator can set bounds to let the GPS algorithm know what determines an attack. The display is fairly straightforward with two input boxes and descriptions next to each. After the operator chooses the values and presses submit, the message will be sent over the RabbitMQ and the Cyber Command display will change accordingly. Additionally, if the inputs entered are not valid entries, the system will let the operator know.

As of the flight test, it was unclear whether having a separate display through a separate URL is insufficient or not. It might be beneficial to have that as a plug-in to the Cyber Command display to minimize workload. Furthermore, the parameter display is very rough and could be built to look more refined; however, there would not be much added practical significance.

5 Conclusions and Future Work

This paper discusses implementation issues associated with the recently introduced System-Aware cybersecurity concept, and discusses a flight evaluation activity of a prototype implementation using those concepts to protect a UAV based surveillance mission. This includes an architectural concept that employs Sentinels for cybersecurity, and the introduction of the term *super-secure Sentinel*. Super-secure smart Sentinels are employed to monitor system functions and assure that they are operating in a manner that is consistent with their design and configuration. In cases where the Sentinels detect inconsistencies, they also designed to automatically determine if the observed differences are due to a cyber-attack. The paper introduces the Sentinel approach for implementation of generally applicable System-Aware architectural solution, resulting in the identification of many design trade-offs related to possible Sentinel implementation. While the trade-offs are conceptually dealt with in this paper, there is a need for related design trade-off studies that address the following Sentinel design trade-off issues

- Multiple Sentinels can be distributed throughout a system, to monitor subsystems and then coordinate on the detection and response to a cyber-attack. Alternatively, a single Sentinel can monitor multiple subsystems and make determinations regarding detection and response. The paper highlights management factors that would contribute to design decisions related to the degree of distribution/centralization.

- Systems that include geographically distributed subsystems, by necessity, must include multiple coordinating Sentinels. Research is required to explore possible architectures for securely integrating Sentinels, and to explore possible information exchange and decision-making architectures required to effectively deal with distribution. Design concepts to support the development of groups of coordinating Sentinels will need to be created to support security for functions that are executed across a system of systems. The paper highlights the integration of UAV-based surveillance, such as imagery collection, which is integrated with a UAV ground-site, UAV autopilot system, and operator control station, as an example of a system of systems for collection and interpretation of imagery. Sentinel designs must be predicated on a systematic approach for (1) identifying potential distributed attacks that require coordination for detection and response, (2) allocating functions to individual Sentinels, and (3) designing the needed secure coordination among Sentinels to detect and respond to such attacks.

- Sentinels can include more functionality related to detection and attack response, but at the expense of expanded and more complex software yielding potential reductions in the security of the Sentinel's implementation. New work is required to develop metrics or design principles that address the relationship between security gained from added functionality and the security compromised from the corresponding additional software and the added complexity of the software design.

- Sentinels need to be secure and can be secured through a variety of techniques. Based upon the experience gained from the UAV-based rapid prototyping project and flight evaluations discussed in section 4.1, the paper provides tangible evidence regarding the ability to advantageously utilize strong but not scalable security techniques, since many (if not all) of a Sentinel’s monitoring and control functions would require little and non-complex software for implementation. Assuming that this observation is representative of a large fraction of the implementation requirements for future Sentinel design patterns, the opportunity to exploit past work on constrained security techniques and to initiate new research on such techniques becomes an increasingly important opportunity.

- Operators will likely be engaged in making what could be critical response decisions in the face of a cyber-attack. The decisions will impact not only current operations, but also must account for intelligence gathering opportunities about the attack and attackers’ objectives, the possibilities for flight replacement instead of system restoration, the consequences of a wrong decision, especially in cases where the fail-over mode is inferior to the primary mode of operation, etc. This points to the need for operations and human factors research regarding decision-making in response to detected (possibly falsely detected) cyber-attacks.

6 Appendix

6.1 Supporting Calculations for GPS System Attack

Proposed Mass Function for FOD (3 components)

Event 1 and Even 8:

$$m_{1,t} = m_{8,t} = \left[\frac{\text{prob}(x-y)}{\max \text{prob}(x-y)}\right] * \left[\frac{\text{prob}(x-z)}{\max \text{prob}(x-z)}\right] * \left[\frac{\text{prob}(z-y)}{\max \text{prob}(z-y)}\right]$$

Event 2 and Event 7:

$$m_{2,t} = m_{7,t} = \left[\frac{1-\text{prob}(x-y)}{\max\{1-\text{prob}(x-y)\}}\right] * \left[\frac{1-\text{prob}(x-z)}{\max\{1-\text{prob}(x-z)\}}\right] * \left[\frac{\text{prob}(z-y)}{\max \text{prob}(z-y)}\right]$$

Event 3 and Event 6:

$$m_{3,t} = m_{6,t} = \left[\frac{1-\text{prob}(x-y)}{\max\{1-\text{prob}(x-y)\}}\right] * \left[\frac{\text{prob}(x-z)}{\max \text{prob}(x-z)}\right] * \left[\frac{1-\text{prob}(z-y)}{\max\{1-\text{prob}(z-y)\}}\right]$$

Event 4 and Event 5:

$$m_{4,t} = m_{5,t} = \left[\frac{\text{prob}(x-y)}{\max \text{prob}(x-y)}\right] * \left[\frac{1-\text{prob}(x-z)}{\max\{1-\text{prob}(x-z)\}}\right] * \left[\frac{1-\text{prob}(z-y)}{\max\{1-\text{prob}(z-y)\}}\right]$$

The probability of the difference of two Gaussian random variables X and Y with means μ_X, μ_Y and standard deviations σ_X, σ_Y respectively is the Gaussian distribution with mean $\mu_X - \mu_Y$ and standard deviation $\sqrt{\sigma_X{}^2 + \sigma_Y{}^2}$. This automatically organizes the variability and accuracy of each sensor. This enables the user to implement cheaper or lighter components with differing accuracy from other components which may improving the scalability and diversity of use of this system.

GLR ALGORITHM

Let μ_1 , the deviation way from μ_0 , be of the form

$$\mu_1 = \mu_0 + \Gamma v,$$

Where Γ is a known vector, and υ is an unknown scalar change magnitude. Substituting this expression for μ_1 allows one to deduce the following expression of the cumulative sum, $S_j^t(v)$

$$S_j^t(v) = \sum_{i=t-W}^{t} \ln \frac{p_{\mu_0 + \Gamma v}(M_j^t)}{p_{\mu_0}(M_j^t)}$$

$$= \sum_{i=j}^{k} v\Gamma' Q^{-1}(m_{j,i} - \mu_0) - \frac{1}{2} v^2 \mu_0 \, \Gamma' Q^{-1} \Gamma$$

These probabilities are assumed to have a Gaussian distribution.

$$\frac{\partial S_j^k(v)}{\partial v} = \sum_{i=j}^{k} v\Gamma' Q^{-1}(m_{j,i} - \mu_0) - (t+1)\Gamma' Q^{-1} \Gamma \, v$$

$$= (t+1)\Gamma' Q^{-1} \left[\frac{1}{t+1} \sum_{i=t-W}^{t} m_{j,i} - \mu_0 \right] - (t+1)\Gamma' Q^{-1} \Gamma v$$

$$= 0$$

So,

$$\hat{v}(k,j) = \frac{\Gamma' Q^{-1} \left[\frac{1}{k-j+1} \sum_{i=t-W}^{t} m_{j,i} - \mu_0 \right]}{\Gamma' Q^{-1} \Gamma}$$

In our algorithm, we have $\Gamma = 1$, and $k - j = W$ our estimate simplifies to

$$\hat{v}_j(t) = \frac{1 Q^{-1} (M_j^t - \mu_0)}{1 Q^{-1} 1}$$

where

$$M_j^t = \frac{1t+1}{}$$

SYSTEMS ENGINEERING
Research Center

Security Engineering Project

Part 1b – System Aware Cyber-Security Application to Unmanned Aircraft Systems (UAS)

Dr. William Melvin, Georgia Tech Research Institute

Georgia Tech Research Institute

Technical Report SERC-2015-TR-036-4 Part 1b

January 31, 2015

Executive Summary

The increasing ubiquity of computerized, automated systems has led to growing interest in the development and application of methods for defense against cyber-attacks. The concern is that vulnerabilities may exist in unmanned autonomous systems that could be easily exploited to compromise the effectiveness of the system. The Georgia Tech Research Institute (GTRI) worked with the University of Virginia on the System Aware Cyber Security project with the goal of developing low-cost, embedded techniques for cyber-attack defense.

The embedded defenses in the System Aware approach are designed to protect the most critical system functions. In designing these defenses the objective is to turn the tables in asymmetric cyber-warfare. Instead of an adversary being able to spend a small amount of money to exploit a system vulnerability and cause great damage, the System Aware designer develops a low-cost defense that forces the adversary to expend significantly greater resources to execute an effective attack. To further keep costs low in applying the System Aware concept, security design patterns should be created that are reusable and that can be repeated at the design level for various types of systems. The System Aware cyber assessment methodology involves identifying the critical system components and determining the likely attach vectors for them. Cyber defenses are then identified and assessed in terms of their potential effect on system performance and the cost to implement.

To illustrate the methodology with a suitable example, the System Aware design process was applied to the GTRI Unmanned Airborne Sensor System (GAUSS). GAUSS is based on an Outlaw ER airframe manufactured by Griffon Aerospace and is equipped with a widely used commercial off the shelf autopilot system and camera gimbal system. A vulnerability assessment of both systems identified several critical aspects of the system including the autopilot parameters (flight plan waypoints, gains, and limits), gimbal commands, the GPS used for navigation, and the data links used for command and control. Three types of attack were selected for development for the demonstration program: an attack that changes the coordinates of the waypoints in the flight plan while masking the aircraft's course change on the operator's display, an attack the corrupts the GPS position data being provided to the camera gimbal for inclusion in the video metadata, and attacks on the camera gimbal commands that make the gimbal appear to malfunction. Methods to detect and defend against these attacks were developed and implemented with a system called the Sentinel. The Sentinel employs design patterns for validating data such as diverse redundancy, data consistency checking, and state estimation. It monitors the serial communications between system components onboard the aircraft and at the ground control station. When a cyber-attack is detected, it notifies the operator through a Cyber Commander interface.

The Sentinel system and the components for carrying out the cyber-attacks were integrated with the GAUSS autopilot system and tested in GTRI's hardware in the loop simulator. The simulator testing allowed the hardware and software configurations to be finalized and thoroughly vetted before the system was integrated into the flight vehicle. After integration and airworthiness qualification, the aircraft was cleared for flight under a certificate of authorization from the FAA for operations at the Early County airport in Blakely, GA. During two flights on Oct. 9, 2014 the Sentinel successfully detected and halted the three types of attacks selected for the demonstration.

Table of Contents

1 Introduction 139

2 System Aware Cyber Security 139

3 General Description of UAS Systems 141

3.1 Autopilots 141

3.2 Camera Systems 142

4 UAS System Vulnerabilities 143

4.1 Autopilot Vulnerabilities 143

4.1.1 Global Positioning System 146

4.1.2 Data Links 147

4.2 Camera System Vulnerabilities 147

4.2.1 High Level Description of Gimbal Attacks 148

4.2.2 Gimbal Attack Specifics 148

4.3 Attacks Selected for Demonstration 149

4.3.1 Waypoint Attack 149

4.3.2 GPS Walk-off Attack 150

4.3.3 Gimbal Command Attacks 150

5 Attack Detection and Mitigation Techniques 150

5.1 Waypoint Attack Defense 150

5.2 GPS Walk-off Attack Defense 151

5.3 Gimbal Command Attacks Defense 151

6 Demonstration Program 152

6.1 Demonstration Platform 152

6.2 Development of the Sentinel 153

6.3 Hardware in the Loop (HIL) Simulation 155

6.4 Tester's Interface 158

6.5 Aircraft Integration 166

6.5.1 System Aware Payload 166

6.5.2 Airworthiness Qualification 166

6.6 Flight Testing 168

6.6.1 Flight Conditions and Restrictions 168

6.6.2 The Test Team 168
6.6.3 Attack Demonstrations 170
7 References.. 175
Appendix A: SerialSpy 176
Appendix B: Gimbal Defenses 179
Appendix C: Schematic Diagrams 182
Appendix D: COA Information 185
Appendix E: Microhard Radio Vulnerability.. 187

List of Figures

Figure 1. COTS Autopilots 141

Figure 2. Typical Flight Control Function Diagram 142

Figure 3. Longitudinal Axis Control Laws 142

Figure 4. Camera Gimbal Functional Diagram 143

Figure 5. External Input Sources for Autopilots 144

Figure 6. GTRI Airborne Unmanned Sensor System 152

Figure 7. GAUSS Autopilot System Block Diagram 153

Figure 8. Onboard Sentinel 154

Figure 9. Ground-based Sentinel 155

Figure 10. Airborne System Architecture 156

Figure 11. Ground System Architecture 156

Figure 12. Baseline HIL Set-Up for GAUSS 157

Figure 13. GTRI HIL Simulator with System Aware Cyber Security Components Installed 158

Figure 14. Tester's Interface Situational Display 159

Figure 15. HUD Indicators 160

Figure 16. Tester's Interface Showing Truth and Attack Positions for the Aircraft 160

Figure 17. Tester's Interface Showing the Defense Position for the Aircraft (in green) 161

Figure 18. Tester's Interface 162

Figure 19. UDP Connections Control (a) Sockets not Connected, (b) Sockets Connected 162

Figure 20. Attack Section of the Tester's Interface 163

Figure 21. ViewPoint Plugin Section 164

Figure 22. Attack Pi Section 164

Figure 23. Masking Section of Tester's Interface (Orbit Waypoint Selection) 165

Figure 24. Masking Section of Tester's Interface (Waypoint List Selection) 165

Figure 25. Payload Chassis 166

Figure 26. Payload installation in GAUSS 166

Figure 27. Test Team inside the GTRI UAV Trailer 169

Figure 28. GPS Walk-off Attack Region 170

Figure 29. GPS Walk-off Attack Position Report 171

Figure 30. Gimbal Command Attack Region 172

Figure 31. Waypoint Attack Flight Plans..174
Figure 32. Simplified Block Diagram Showing SerialSpy Usage in a UAV Sentinel....................176
Figure 33. Gimbal Command Authentication 180
Figure 34. Battery Protection and Payload Kill Circuit..183
Figure 35. System Aware Payload Schematic 184
Figure 36. COA Area of Operations..185
Figure 37. COA Area of Operations in Detail 186
Figure 38. Microhard MHX2420 radio module 187
Figure 39. Spectrum Plot of Beacon Dwell Centered at 18 kHz 188
Figure 40. Zoomed In Plot of Beacon Dwell Centered at 18 kHz 189
Figure 41. Overlaid Plot of FM Demodulated Audio (Green) and Assigned Symbols (Pink) of Beacon Dwell 189
Figure 42. Example decoding process for a single over-the-air dwell 190
Figure 43. Interpreted bit field of beacon dwell RF packet example.......................................191
Figure 44. Data Raster of Master Transmission of Data 192
Figure 45. Example RF packets of slave dwell bit structure with characters A-I being transmitted 194

1 Introduction

Several incidents in the past few years have increased concerns about the vulnerability of the U.S. UAS fleet. In one incident, reported in December 2009, militants in Iraq used $26 off-the-shelf software to intercept live video feeds from U.S. Predator drones, potentially providing them with information they need to evade or monitor U.S. military operations (Gorman, Dreazen, & Cole, 2009). In another incident, occurring on December 4, 2011, a Lockheed Martin RQ-170 Sentinel UAS was captured by Iranian forces near the city of Kashmar in northeastern Iran. The Iranian government claimed that the UAS was commandeered by its cyber warfare unit although U.S. officials dispute this claim. Both of these incidents involved high value assets that would be expected to have numerous security features to prevent cyber-attacks. The less expensive and far more numerous systems such as the tactical UASs and small UASs may not have significant cyber security measures and would thus be far more vulnerable to a cyber-attack. The ubiquity of these systems on the modern battlefield heightens the concern over their vulnerability. This provides ample motivation for applying the principles of cyber security asymmetry to develop low cost protection features that greatly increase the effort and cost that an adversary must expend to exploit the system's vulnerabilities.

Cyber-attacks on unmanned systems may come from a number of different sources including attacks through the supply chain of components used in the system, insider attacks from people with access to the system, malware attacks from corrupted software sources, and attacks through the command and control data link. The goal of System Aware Cyber Security is to develop low-cost, embedded techniques for cyber-attack defense (Jones & Horowitz, 2012). The approach is to create reusable system aware security designs that can serve as patterns for solutions and that can be repeated at the design level for various types of systems.

This report illustrates how the System Aware approach may be applied to an actual unmanned aircraft system. The research program was conducted in two phases. In the first phase a widely used commercial autopilot and the camera gimbal system were examined for vulnerabilities and several potential attack vectors were identified. A subset of these attacks was then selected for development along with methods to detect and defend against these attacks. In the second phase of the program the hardware and software required for demonstrating System Aware on a UAS was developed. The system was thoroughly tested in simulation then demonstrated in a flight test on October 9, 2014. This report documents both phases of the System Aware program.

2 System Aware Cyber Security

Developing low-cost, embedded techniques for cyber-attack defense has been the goal of a research program sponsored by the Systems Engineering Research Center (SERC) and the Department of Defense. The research team, consisting of the University of Virginia (UVa) and the Georgia Tech Research Institute (GTRI), developed a novel cyber security concept for embedding defense mechanisms within highly automated systems such as UAS. This concept is referred to as System Aware cyber security.

System Aware operates at the system application layer to provide security inside the system network; i.e. behind the perimeter protection provided for the whole system. While networked systems typically have some measure of protection at their points of access, these defenses can be breached. In addition, attacks by trusted insiders or through the supply chain of the system components are potential avenues that can bypass network perimeter defenses.

The embedded defenses in the System Aware approach are designed to protect the most critical system functions. In designing these defenses the objective is to turn the table s in asymmetric cyber warfare. Instead of an adversary being able to spend a small amount of money to exploit a system vulnerability and cause great damage, the System Aware designer develops a low-cost defense that forces the adversary to expend significantly greater resources to execute an effective attack. To further keep costs low in applying the System Aware concept, security design patterns should be created that are reusable and that can be repeated at the design level for various types of systems.

The design patterns that may be used for cyber defense include, but are not limited to:

- Diverse redundancy by using diverse redundant components in critical subsystems to restore functionality in the event one component is disabled
- Diverse redundancy with voting by using diverse redundant components, comparing their outputs, and looking for differences outside of the expected error bounds to detect an attack. In a triplex system, two components that are in agreement can detect the failure of the third component and eliminating the infe cted component.
- Configuration hopping by physically or virtually modifying the configuration of the hardware/ software components in systems dynamically to provide a moving target defense
- Data consistency checking for data integrity and operator display protection
- Physical confirmations of digital data for data integrity
- Use of analog components for diversely redundant solutions
- State estimation using existing information in the system to estimate other state variables in the system and inferring the overall state of the system

To develop effective, low cost System Aware design patterns researchers at UVa have developed the relational System-Aware cyber assessment methodology. The process consists of the following six steps:

1. identify the critical system components for a particular system
2. identify the possible paths to attack those components
3. determine which of those attack paths would be most desi rable to an adversary
4. identify possible cyber security defenses against those attacks as well as evaluate the impacts of those defenses on the attacker
5. assess the effects on system performance of potential defenses
6. estimate the security trade-offs of the various architectural solutions

Details of this methodology are provided in our Phase I Final Report. This is the methodology that was applied to developing System Aware design patterns for UAS as described in this report.

3 General Description of UAS Systems

3.1 Autopilots

The current proliferation of unmanned aircraft has been greatly aided by the availability of several commercial off-the-shelf (COTS) autopilots. Among the most popular systems are the Piccolo family from Cloud Cap Technology, the MP2x28 series from MicroPilot, and the Kestrel autopilot from Lockheed Martin (Figure 1). These autopilots include a number of sensors such as a GPS receiver for geo-location, an inertial measurement unit (IMU) consisting of accelerometers and rate gyros, and pressure transducers to measure airspeed and altitude. They also include a microprocessor that performs signal processing and hosts the feedback control laws used to stabilize and steer the aircraft. A digital spread spectrum radio link enables the ground operator to send flight plans to the autopilot, adjust system parameters, and receive telemetry and status messages.

Piccolo II MP2128 Kestrel

Figure 1. COTS Autopilots

Figure 2 shows the functional structure of a typical autopilot system. The system has two primary components: the autopilot onboard the aircraft and the ground control station consisting of a transmitter/receiver, a pilot console, and a computer hosting the operator interface software and serving as a display. In the autopilot, the guidance function receives the flight plan in the form of a series of waypoints from the operator. Based on the aircraft's current position and heading and the location of the next waypoint, the guidance function determines the heading rate and climb/dive rate to steer the aircraft to the waypoint. The control laws generate commands to the flight control actuators to stabilize and control the aircraft's motion and manage its power setting. An example of a control law structure for the longitudinal axis is presented in Figure 3. In the innermost loop pitch rate is fed back to the elevator to provide damping and improve the aircraft's stability. In the next loop, pitch attitude is fed back to stabilize the aircraft's phugoid mode (a low frequency airspeed-altitude oscillation). The outermost loop could employ airspeed feedback to control the airspeed (as shown) or, alternatively, altitude and altitude rate could be fed back to control the attitude. Limiters are used to prevent saturation of the control loops and saturation of the actuator commands. The gains and limits are initially set by the flying qualities engineer based on simulation and further tuned through flight testing.

The autopilot sensor suite includes a GPS receiver for determining the aircraft's location, ground speed, track angle, and geodetic altitude. The IMU measures the aircraft angular rates about the pitch, roll, and yaw axes, as well as the accelerations along these axes. Pitot and static pressure transducers measure the aircraft's airspeed and altitude. A magnetometer is used to measure the aircraft's yaw angle with respect to magnetic north. Other sensors are typically included to measure engine speed and temperature and the aircraft's height above ground level. Usually the signals from the sensor suite are processed through a Kalman filter to minimize the effects of noise and biases and to estimate states that are not measured directly, such as the aircraft's pitch and roll angles. The resulting aircraft state vector is fed back to the control laws for stability and control and to the guidance function to compute the outer loop flight path commands. The information is also telemetered to the ground control station to inform the operator of the aircraft's trajectory and status.

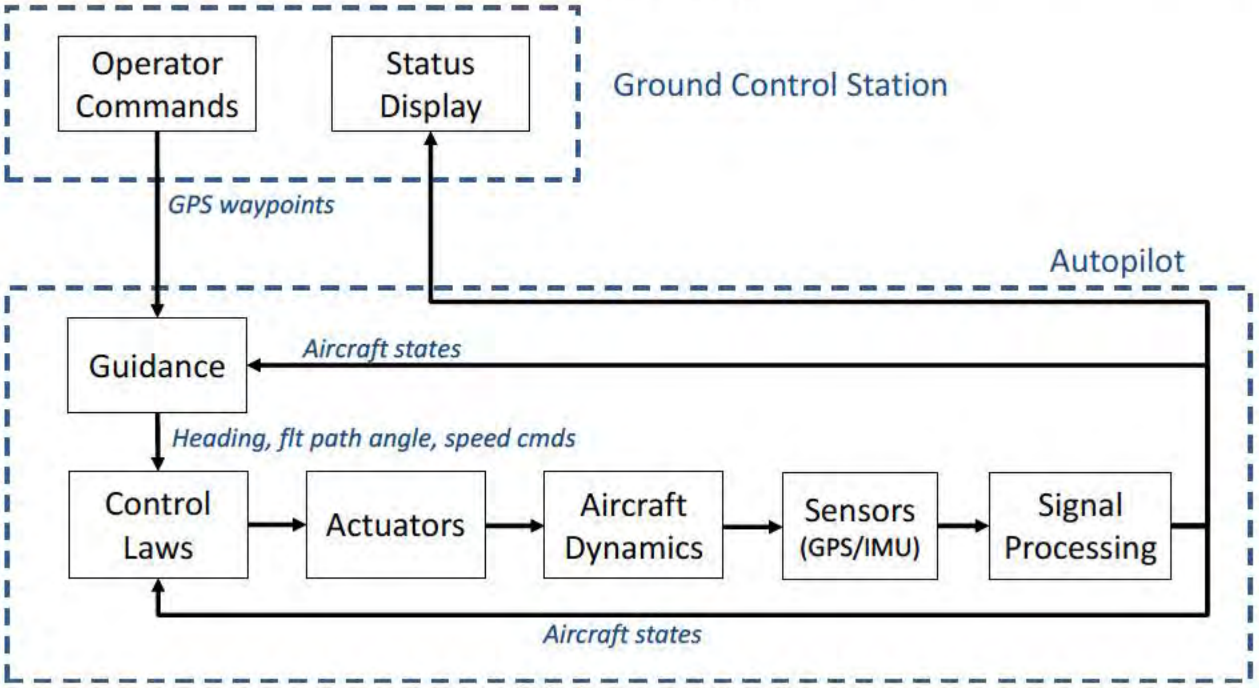


Figure 2. Typical Flight Control Function Diagram

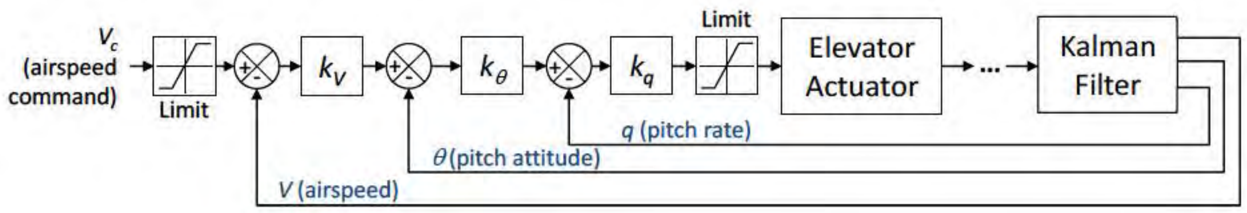


Figure 3. Longitudinal Axis Control Laws

3.2 Camera Systems

UAS usually carry some type of payload consisting of sensors for monitoring the external environment. Most often the payload sensor is a video camera system that relays full motion video back to a payload operator collocated with the UAS operator. Figure 4 presents a generic functional diagram for a camera gimbal system. Through an interface at the ground control station the camera operator is able to send commands to control the orientation of the camera, select operating modes such as target tracking, and control camera settings such as the zoom level. A processor within the camera gimbal executes control algorithms to point the camera in the desired direction while compensating for the aircraft's motion. This motion may be measured by a GPS/IMU sensor suite within the gimbal or be provided by the autopilot's GPS/IMU. The gimbal states are also relayed to the operator to provide him awareness of the gimbal's orientation and status.

The video from the camera may be further processed to stabilize the scene and add text overlays containing information about the image such as the coordinates of a target. More sophisticated camera gimbal systems include tracking algorithms that allow the camera operator to designate a portion of the image and have the camera system automatically track it. These algorithms may be executed by a processor within the camera gimbal or in the ground control station.

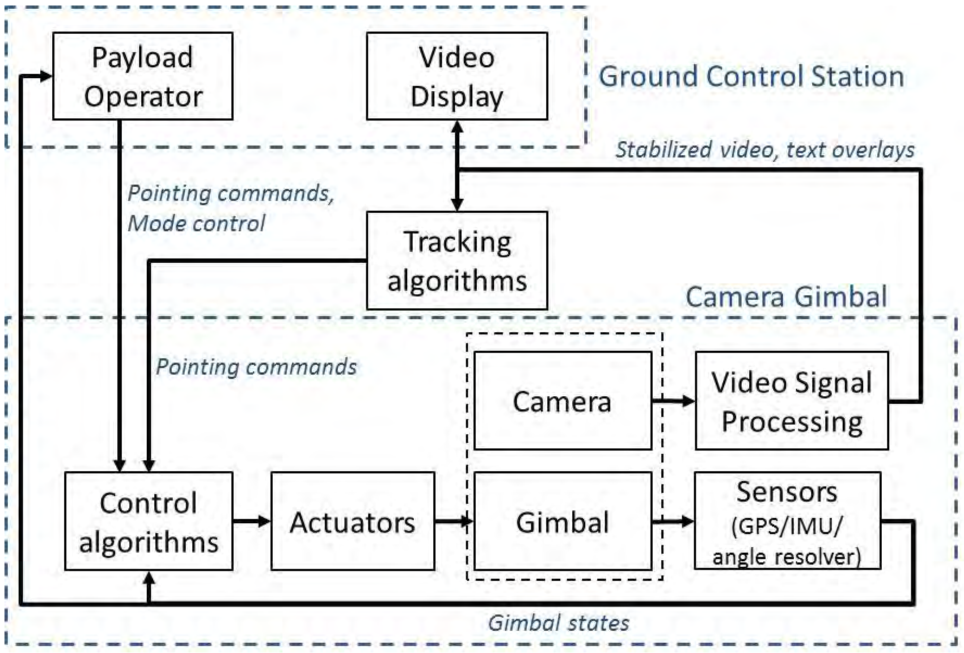


Figure 4. Camera Gimbal Functional Diagram

4 UAS System Vulnerabilities

4.1 Autopilot Vulnerabilities

Today's COTS autopilots are extremely versatile and can be easily tailored for a variety of applications. This versatility, however, also provides a number of avenues for exploitation that could be used to adversely affect the behavior of the aircraft. Figure 5 illustrates the fact that many of the components that make up an autopilot have inputs from external sources. Guidance receives waypoint lists, flight path commands, and mode commands, such as flight termination, from the ground control station. The control laws have a number of gains and limits that can be changed via the ground control station. The results of changing these parameters could be subtle, such as an annoying inability to maintain a desired altitude or flight path, or catastrophic, such as flight into terrain or an engine shutdown. For an example of how one small change could adversely affect a UAS's behavior, consider the longitudinal axis control laws shown in Figure 3. Simply setting the pitch rate gain (k_q) to zero would effectively lock out control to the elevators. This could also be accomplished by setting the actuator command limiter to zero. Nearly all of the parameters in the flight control laws can be changed by the operator through the radio link while the aircraft is in flight. The details of what parameters can be changed, what impact they have on the flying qualities, and what data security measures are employed are particular to each autopilot make and model.

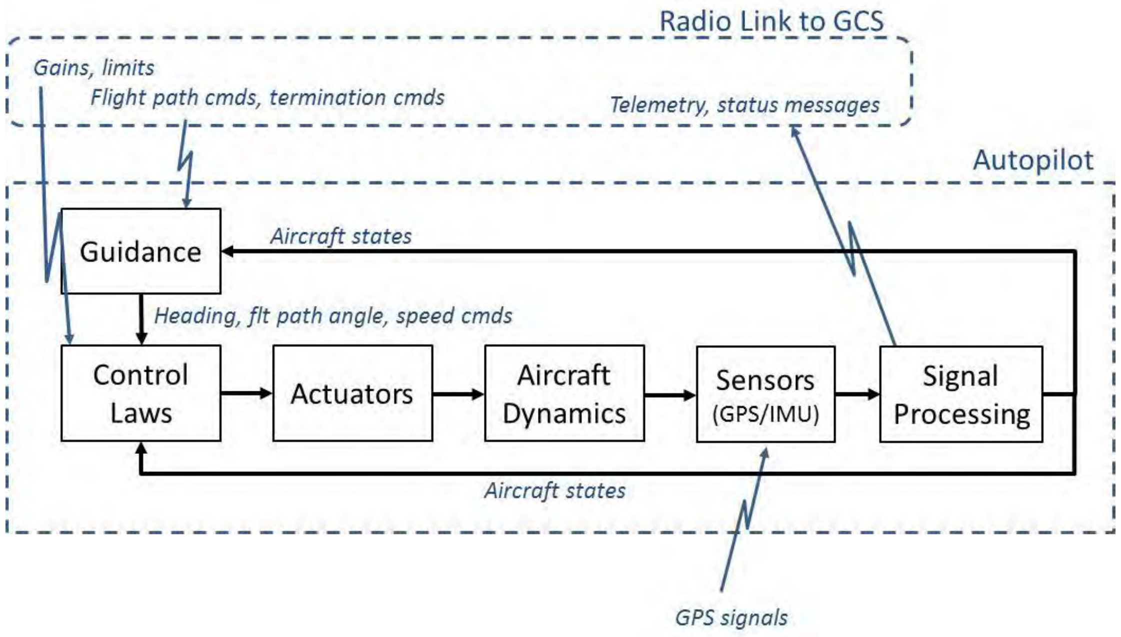


Figure 5. External Input Sources for Autopilots

To illustrate the significance of this issue, Table 1 presents a partial list of the parameters that can be changed in the Piccolo II autopilot while the aircraft is in flight. The list is color coded to indicate the potential severity of changes to these parameters. Exploitation of items colored in red will cause the loss of the aircraft. For example, certain gains and limits could lock out control of the aircraft if they are set to zero. Items colored in orange could possibly cause the loss of the aircraft, depending on its current state. For instance, raising the bank angle limit above 30 degrees could potentially allow the aircraft to enter too tight of a turn causing the wing to stall and resulting in loss of control. Items color coded in yellow could degrade the aircraft's flying qualities by preventing it from holding trim, for example. Items not color coded need to be assessed through simulation to understand their potential impact on the flying qualities and control of the aircraft.

Table 1 also indicates the security measures for the autopilot parameters. In many cases, changing a parameter value requires the operator to enter a password before the change is accepted. This is an example of an inexpensive approach to ensuring data integrity. As a result, the more likely avenue of attack will be on parameters that are not password protected.

Table 1. Piccolo Parameters

Group	Parameter	Purpose	Possible exploitation	Protection
Navigation	GPS signals	geolocation	spoofing timing signals could result in a false position solution	
Actions	Abort	action depends on current aircraft state		
	Engine On	allows user to enable or disable engine		not protected
Flight plan	waypoint coordinates	control flight path of aircraft	altitude can be zeroed, waypoint can be moved	not protected, but changes reflected on operator display
Command limits	dynamic pressure limit	limit aircraft airspeed		password protected
	pressure altitude limit	limit aircraft altitude		password protected
	bank angle limit	limit aircraft turn rate	extending limit beyond 30° could cause aircraft to stall	password protected
Output limits	aileron travel limits	limits autopilot cmds to ailerons	setting limits to 0 could shut down actuator cmds	password protected
	elevator travel limits	limits autopilot cmds to elevator	setting limits to 0 could shut down actuator cmds	password protected
	rudder travel limits	limits autopilot cmds to rudder	setting limits to 0 could shut down actuator cmds	password protected
	throttle travel limits	limits autopilot cmds to throttle	setting limits to 0 could shut down actuator cmds	password protected
	flap travel limits	limits autopilot cmds to flaps	setting limits to 0 could shut down actuator cmds	password protected
Mission limits	pilot timeout	controls switch to autopilot when no manual input	setting limit to 0 will block possibility of manual flight	password protected
	comm timeout	controls when Lost Comm Waypoint plan is executed		password protected
	GPS timeout	defines the amount of time for the aircraft to continue without a GPS solution		password protected
Lateral gains	roll rate gain	roll damping	setting this gain to zero will disable roll control	password protected
	roll rate integral gain	used to trim errors in the ailerons	setting gain to zero will disable ability of	password protected

			autopilot to trim out aileron errors	
Longitudinal gains	energy rate error integral to throttle	gain on integral of energy rate error to throttle	setting gain to zero will inhibit throttle trim	password protected
	altitude error to altitude rate gain	gain on altitude error to commanded altitude rate	setting gain to zero will inhibit altitude control	password protected
	Altitude rate error to Z acceleration cmd	Gains sets the bandwidth for the desired vertical rate	setting gain to zero will inhibit altitude control	password protected
Limits	Aileron Max	Maximum aileron output	Setting limit to zero will lock out aileron control	password protected
	Elevator Min	Minimum elevator output	Setting limit to zero will lock out aileron control	password protected
	Elevator Max	Maximum elevator output	Setting limit to zero will lock out aileron control	password protected

4.1.1 Global Positioning System

Another potential vulnerability of most autopilots is their heavy reliance on the Global Navigation Satellite System (GNSS), in particular the Global Positioning System (GPS), for navigation. This dependence on GPS leads them to be susceptible to jamming or spoofing attacks. Although jamming is much more straightforward to accomplish, it can lead to the terminal flight of the aircraft and will be immediately obvious to the operator of the vehicle. Spoofing can be covert and is capable of controlling the vehicle's flight through the attack.

Adversaries developing GPS spoofing attacks can take advantage of the open knowledge of the GPS satellite messages. In civil GPS systems, the pseudo-random number code is defined in openly available GPS interface documents. The repetition every millisecond allows the message to be easily predicted. If a spoofing signal can broadcast this message with a slightly varying phase, then the receiver's coordinates can be altered, allowing the attacker to commandeer the aircraft. This was demonstrated by Kerns[2] at the University of Austin, Texas in 2012 when a rotorcraft UAV was covertly captured via a spoofed satellite signal and subsequently controlled by the spoofed UAV coordinates.

Several methods of anti-spoofing are present in modern day GPS sensors. Jamming-to-noise monitoring trips an alarm whenever the power level in a given band reaches a certain dB higher than the background noise in typical conditions. Frequency unlock monitoring is an effective deterrent to spoofing because it requires the attacker to have specific data about the aircraft's velocity. Another common method is innovations testing. This compares the position-velocity output of the GPS to the state estimator in the Kalman filter. If the two values disagree then an alarm is triggered. The University of Austin demonstrated that even with these methods, the system is still vulnerable and the attacker can still covertly bypass these prevention techniques (Kerns, Shepard, Bhatti, & Humphreys, 2014).

One prevention method presented in the System Aware program is triple modular redundancy with dissimilar systems. Using various GPS sensors and a voting scheme, the vulnerability in one system can be compensated by the other two. The best situation would be to have GNSS systems using different satellite constellations, e.g. one GPS, one Galileo, and one GLONASS system. The variation in the carrier frequencies and messaging protocols for each of these systems ensures that all three systems could not be spoofed at the same time.

4.1.2 Data Links

Yet another avenue of attack on autopilot systems is through their command and control link. A recent study by Hartmann and Steup (Hartmann & Steup, 2013) looked at several types of data links currently being used in UAVs and assessed their susceptibility to cyber-attack. The links included the Tactical Common Data Link (TCDL) in the Ku band, line of site (LOS) communications in the C band, and in the WiFi bands using the IEEE 802.11 standard. Another commonly used band is the industrial, scientific, and medical (ISM) band at 900 MHz and 2.4 GHz. The command and control link for the Piccolo II autopilot used in this project is the MHX-910/2400 frequency hopping radio from Microhard Systems Inc. GTRI analyzed this radio system to determine how the link might be compromised. Our initial assessment is presented in Appendix E.

4.2 Camera System Vulnerabilities

UAVs are predominantly used as ISR platforms carrying sensor payloads such as EO/IR cameras, synthetic aperture radar, signals intercept systems, and others. As a result, sensor technology is evolving quickly with new sensor systems being developed for all classes of UAVs. However, in the push to quickly field these new sensor suites and take advantage of their capabilities, cyber security is sometimes neglected. This creates an opportunity for an attacker to compromise a mission by exploiting weaknesses in the payload security; e.g., an attacker could degrade or deny the payload service or spoof the information coming from it.

Currently, video cameras are the most prevalent sensors used on UAS. Camera gimbal systems share some of the same vulnerabilities as autopilots such as their dependence on GPS for position data. They also typically use the autopilot's command and control link to pass messages from the operator's station through the autopilot to camera gimbal. Exploitation of these vulnerabilities could significantly compromise the effectiveness of the UAS's mission without the need to attack the autopilot system itself. For example, a cyber-attack could corrupt the GPS position data used by a camera gimbal system for determining the geolocation of the video image. When the erroneous position data is included in the video metadata any targeting value is lost. This type of attack could be difficult to detect since it does not affect the behavior of the aircraft. In a similar vein, cyber-attacks on the gimbal commands can make it appear that the system is malfunctioning and thus reduce the mission effectiveness. Examples of these types of attacks include malicious commands sent to the camera gimbal that interfere with gimbal deployment and object tracking. Because these types of attacks are subtle, the operator may not realize that the system is under attack and just chalk it up to a random malfunction.

To investigate methods for preventing, detecting, and countering potential cyber-attacks against camera gimbal payloads, the GTRI studied potential cyber-attacks and corresponding cyber security solutions for the TASE 150 camera gimbal system on its GAUSS UAV. The TASE

150 is a member of the popular and widely used family of TASE camera gimbal systems developed by Cloud Cap Technology.

4.2.1 High Level Description of Gimbal Attacks

In order to determine the simplest vector to compromise the TASE camera gimbal, GTRI analyzed the specifics of the TASE gimbal, the ViewPoint ground station software (used to view the video), and the communications protocol used to issue commands to the gimbal as well as receive status updates from the gimbal. This analysis revealed that the simplest attack vector would be to cause a denial of service or degradation of service by sending malicious, unauthorized commands to the gimbal from a malware exploit running on the operator interface machine (i.e., the machine hosting the Piccolo Command Center (PCC) and ViewPoint).

This type of attack is possible because it is assumed that the source for all gimbal commands can be trusted. This means that as long as attackers can communicate with the gimbal, they can have it execute any command they want. In addition, there are multiple commands that can potentially be exploited by an attack to cause a denial or degradation of service. Together, these factors suggest this path of attack.

The attack vector chosen for this study embeds a malicious exploit into ViewPoint. Embedding the malicious exploit is made possible by the open architecture of the ViewPoint and PCC software that allows developers to create plug-in software modules for added functionality. In addition, the PCC and ViewPoint allow users to go online and download maps and aerial imagery from several different map databases. No particular security measures are in place for users downloading maps onto the machine hosting the PCC or ViewPoint. Together these features provide a potential attack vector.

An alternative attack vector was considered that required communicating with the gimbal directly from a rogue wireless command tower. However, it was determined that the simplest solution would be to use the already established communication channel. In addition, solutions designed to detect malicious data sent from the operator interface should also be able to detect malicious data sent from an alternate source.

4.2.2 Gimbal Attack Specifics

The attack is an exploit embedded into ViewPoint that sends malicious data to the gimbal. The data will be unauthorized but properly constructed command packets designed to cause a denial or degradation of service. The exploit has the ability to construct the command data, compute the checksum, and send it to the gimbal. In addition to sending malicious data, the exploit can also produce non-malicious data at random intervals to attempt to hide the malicious data. The following are commands that could be used for a degraded or denial of service attack.

4.2.2.1 0x00 / 0x43: Extend/Retract Gimbal

By issuing commands to retract the gimbal during critical points in the mission, an attacker can cause the loss of a significant amount of information. By continuously issuing the command to retract the gimbal an attacker can cause a complete denial of service of the payload.

4.2.2.2 0x00 / 0x70: Disable Motor Driver

As with the Retract Gimbal packet, this command can cause a similar denial of service by interfering with the operator's ability to steer the camera gimbal.

4.2.2.3 0x00 / 0x80: Gimbal Command

This command controls the location in which the gimbal is pointed. Pointing the gimbal away from the target can cause a denial of service. Random or erratic movement of the gimbal may cause the camera operator to assume a technical malfunction has occurred and recall the UAV.

4.2.2.4 0x00 / 0x40: Gyroscope Zero

This command sets the zero of the gyroscope on board the TASE gimbal. The gimbal documentation warns that the operator should not issue this command while the gimbal is in motion. Doing so may cause the gyroscope to be calibrated improperly, causing a degradation of service that would be difficult to fix mid-flight. This may force a recall of the UAV. The full extent to which this would affect performance has not yet been determined.

4.2.2.5 0x28 / 0x00: User Warning Packet

This packet is sent to the ViewPoint software instead of the gimbal. The software will display an error or warning message to the operator, which may be used to social engineer the operator into aborting the mission or taking other actions based on false information.

4.3 Attacks Selected for Demonstration

Three types of cyber-attacks were chosen and fully developed for the demonstration program: an attack that changes the list of waypoint in the autopilot's flight plan, an attack that corrupts the GPS position, and attacks on the camera gimbal that appear as gimbal control malfunctions. The demonstration program proceeded from the premise that the defense against a particular type of cyber-attack should be general enough that it is largely agnostic to the specifics of how the attack was implemented. In other words, regardless of whether the attack was caused by a corrupted avionics component, a hacked data link, malware infecting a ground control station, or some other means, the defenses should be able to detect and halt the attack. Therefore, rather than trying to hack into the various autopilot system components and install the attack software directly on them, we used several Raspberry Pi single board computers (SBCs) to host the attack software. The devices were connected to the serial communications links between components where they could intercept and modify message traffic to effect the various attacks. The specific attacks that were implemented are described in the following subsections.

4.3.1 Waypoint Attack

The waypoint attack changes the waypoints in the autopilot's flight plan causing it to fly a different trajectory from the one intended by the operator. To execute the attack, the tester sends a new list of waypoints via Ethernet to a Raspberry Pi onboard the aircraft that is connected to one of the autopilot's serial communication ports. The attack Pi pushes the new list of waypoints to the autopilot through the autopilot message stream. Since the autopilot sends the updated waypoint list to the operator's station, the change would normally be readily apparent. However, researchers at UVa developed a companion attack at the ground control station that would mask the change to the waypoint list as well as mask the location of the aircraft as it flies the new flight plan. As a result, during the attack the operator's station displays the aircraft where he expects it to be.

4.3.2 GPS Walk-off Attack

When the aircraft enters a pre-defined geographic region, the GPS location data that is sent from the autopilot to the camera gimbal is corrupted with a slowly increasing bias. The corrupted GPS data is included in the video stream as metadata that provides the coordinates of the area being viewed by the camera. As a result of this attack, the video imagery has erroneous position data associated with it and it loses all value for ISR and targeting purposes. The GPS walk-off attack is implemented with a Raspberry Pi SBC located on the serial link between the autopilot and the camera gimbal. When the attack is disabled, serial communications pass between the autopilot and camera gimbal unaltered. When the attack is underway, the attack Pi intercepts the GPS data packet, adds a growing position bias to it, and passes it on to the camera gimbal.

4.3.3 Gimbal Command Attacks

These attacks use the gimbal command set to make the gimbal appear to malfunction. When the aircraft enters a predefined geographic region, spurious gimbal commands are issued to the gimbal causing it to retract or slew the sensor point of interest (SPOI) upwards. The gimbal retract command is issued whenever the aircraft is in the attack zone and the camera operator tries to engage the camera's tracking function. Similarly, when the SPOI attack is enabled the camera will slew upwards when the camera operator engages the gimbal's SPOI function. These attacks are caused by malware hosted on the camera gimbal operator's station. The software monitors the operators inputs and uses them to trigger the retract and SPOI attacks.

Implementing the attacks through separate SBC devices greatly mitigates the risk in flight testing. Because the flight critical avionics components remain unchanged and the attack software is hosted on separate SBC devices, these devices can be shut down to restore normal functionality in the event of an in-flight emergency. Another measure that was taken to reduce risk was how the GPS attack was implemented. Instead of attacking the GPS data used directly by the autopilot system for navigation (which could be extremely dangerous), the GPS data provided to the camera gimbal was attacked. This allowed us to demonstrate the ability of the system to detect and defend against GPS attacks without put the aircraft at risk.

5 Attack Detection and Mitigation Techniques

5.1 Waypoint Attack Defense

The spoofing of waypoints to the autopilot is possible because of the lack of authentication between the autopilot and the command's source. The prevention of this attack is accomplished by using a Python based key logger and image capture on the ground station to record the operator's inputs. Onboard the aircraft the autopilot commands are monitored for changes to the waypoint list. These changes can then be sent to the ground station where they are compared to data from the key logger. If the command is not found within the logged data, then it must not have originated from the ground station. In this event, a flag can be sent to the operator notifying him of the attack.

5.2 GPS Walk-off Attack Defense

To mitigate or eliminate this attack a triple modular redundant (TMR) system of GPS receivers is used. The autopilot GPS and two additional GPS receivers are used to vote on the aircraft's position. A threshold is set in the TMR system which triggers a notification if the GPS error of one of the systems was greater than a specified limit. The TMR system can maintain functionality by using the GPS position report from the two systems in agreement.

5.3 Gimbal Command Attacks Defense

Degradation and denial of service attacks are possible because the gimbal trusts the sender of any commands that it receives. To prevent this type of attack, the system should be able to evaluate any command it receives to determine its validity.

Implementing defenses by modifying how the ViewPoint software issues commands or how the gimbal responds to them was not practical because these are commercial products and the source code is not available. However, it is possible to place a piece of in-line hardware or software on the UAV that receives the command packet before the gimbal and decides whether or not to forward it to the gimbal based on mission conditions.

Several methods can be used to make decisions on the validity of gimbal commands. One method to help catch unauthorized commands is to implement an authentication scheme, possibly by appending a cryptographic signature to messages sent from the ViewPoint software to the gimbal. However, this will not protect the gimbal system from the case in which a malicious agent has compromised the PCC. Depending on the degree of compromise, the malicious agent could still be able to send messages that the gimbal would consider authentic.

In a similar manner, providing additional authentication to commands capable of causing damaging effects would be helpful but not sufficient. An attacker who has not fully compromised PCC to the point of recovering the cryptographic key would be halted by such a defense, but further compromises may render this ineffective.

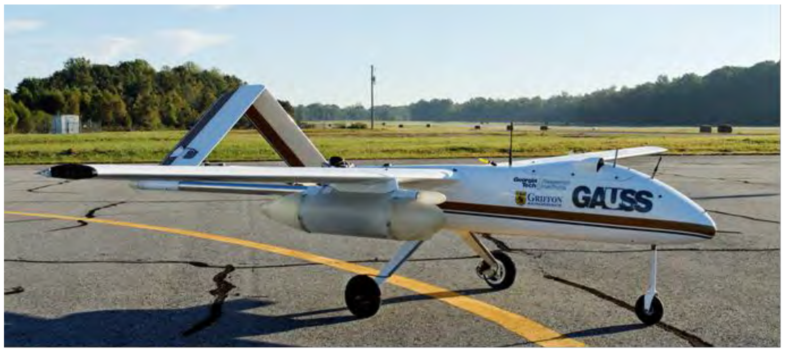
To protect from compromises of PCC the system should be able to judge the legitimacy of commands. To do this a run-time analysis can be performed to determine whether or not executing a command makes logical sense based on mission context. Determining the mission context is a form of state estimation in that aircraft states can be used to infer indirectly what stage of the mission the UAS is in. For example, if the system received a command to retract the gimbal while it is in a pre-specified area of interest, an intelligent decision would be to not immediately trust the command and attempt to verify its authenticity. In addition, authorized operators should be able to issue whatever commands they need, so there must be an override capability to verify that traditionally illogical commands are in fact legitimate. Appendix B provides details on the defenses that were developed for the TASE camera gimbal.

6 Demonstration Program

The demonstration program was conducted during the second phase of the System Aware project. The demonstration platform was provided by GTRI and the System Aware concept was implemented on hardware and in software for this platform. The following subparagraphs describe the demonstration effort.

6.1 Demonstration Platform

The UAS selected demonstration purposes was GTRI's Airborne Unmanned Sensor System (GAUSS) (see Figure 6). GAUSS is based on an Outlaw ER airframe manufactured by Griffon Aerospace and is equipped with a Piccolo II autopilot system and a TASE 150 camera gimbal system, both manufactured by Cloud Cap Technology™. The Piccolo II's datalink employs a frequency hopping radio from Microhard Systems Inc. operating at 910 MHz. GAUSS is equipped with a secondary radio link at 450 MHz for additional reliability. Communications through the radio datalink with the ground control station are managed with several message streams such as a pilot in the loop stream for manually piloted operation, an autopilot stream to communicate with the Piccolo avionics, a gimbal stream to control a camera gimbal, and a payload stream for serial communication with a payload. These bi-directional streams are multiplexed onto the wireless link. At either end of the RF link is a serial RS-232 port to connect to either the onboard processor in the autopilot and the operator interface computer at the ground control station. GAUSS can also be controlled manually from a wireless joystick console broadcasting at 2.4 GHz.



Parameter	Value
Max Takeoff Gross Wt.	150 lbs
Wing span	16 ft
Payload capacity	35 lbs
Endurance	1-2 hrs
Cruise speed	75 kts
Installed power	16.5 hp

Figure 6. GTRI Airborne Unmanned Sensor System

As shown in Figure 7 the autopilot has a several serial communication ports and discrete I/O lines for interfacing with payloads and other components. One of the serial COM ports is occupied by the 450 MHz radio. Another COM port is used to communicate with the TASE 150 camera gimbal. A third COM port is connected to a discrete magnetometer that provides a heading reference for the navigation filter that is independent of the GPS receiver. Two of the general purpose I/O (GPIO) lines are dedicated to the manual pilot control at 2.4 GHz. The two receivers are oriented orthogonal to each other to ensure reliable communications with the pilot console on the ground. A level shifter board adjusts the voltage level between the receivers and the GPIO lines. Another GPIO line is used to control a kill switch on the payload power supply board. This arrangement allows the GAUSS operator to kill power to the payload systems from the Piccolo operator interface.

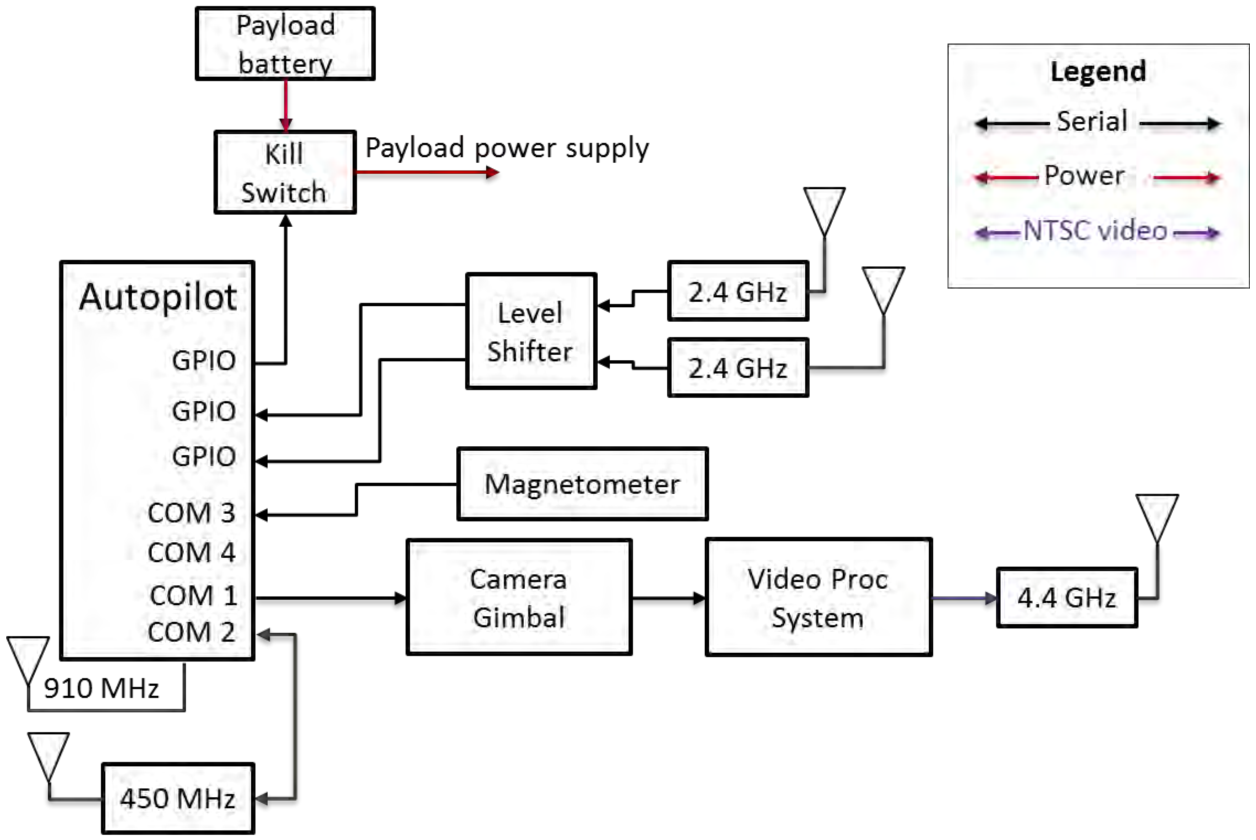


Figure 7. GAUSS Autopilot System Block Diagram

6.2 Development of the Sentinel

To implement the System Aware cyber defenses for the UAS application with GAUSS, UVa and GTRI developed the concept of a Sentinel which monitors the serial communications between system components and looks for anomalies. Onboard the aircraft, the Sentinel comprises a microcontroller called the SerialSpy, three Raspberry Pi single board computers (SBCs), a serial communications splitter, an Ethernet switch, and a SiCore secure communications interface board called the Shield II. Figure 8 shows how the Sentinel interfaces with the existing autopilot and camera gimbal system. In implementing the demonstration system we used two of the serial COM ports on the autopilot which required the magnetometer to be disconnected.

The SerialSpy shown in Figure 8 monitors the communications between the autopilot and the gimbal. A detailed description of the SerialSpy is provided in Appendix A. The SerialSpy defends against GPS attacks by checking the autopilot's GPS position data against validated position data provided by the SiCore Shield II. Three additional Raspberry Pis (referred to as Air Sentinels and labeled AS1, AS2, and AS3 in the figure) are included as part of a triple redundant voting scheme to validate GPS position data. Two of these Pis have independent GPS receivers while the third uses the autopilot GPS data acquired via the Ethernet link. If the GPS position data does not match the validated data within some error bound, an attack is indicated and the SerialSpy passes on to the gimbal the validated position data provided by the Sentinel. The SerialSpy also monitors and validates the gimbal commands that are relayed through the autopilot to the camera gimbal. The SerialSpy disables commands that defy normal operating procedures such as a gimbal retract command during a critical portion of the mission.

The SiCore Shield II monitors serial inputs to the autopilot and checks the message streams for changes to autopilot parameters such as the waypoints in the flight plan. A passive serial splitter is used to tap into the communications between the autopilot and other devices connected to it. As shown in Figure 8 the splitter is located on the line to the COM3 serial port. This is the serial port that was used to inject malicious changes to the flight plan during the demonstration. In general a tap would be placed on each of the communication lines leading into the autopilot. The Shield II also manages the voting scheme used to validate data in triple redundant systems. The version of the Shield used for the flight demonstration is based on SiCore Technologies' Shield coprocessor which incorporates a number of security and anti-tamper measures.

The ground-based portion of the Sentinel is shown in Figure 9. A Raspberry Pi SBC (the Ground Sentinel in the figure) works in conjunction with the onboard Sentinel to detect attacks that affect autopilot parameters such as the waypoints in the flight plan. The onboard Sentinel monitors the inputs received at the autopilot while a ground-based Sentinel monitors the operator's inputs by analyzing the logged keystrokes. When a command such as a change to the waypoint list is received by the autopilot, the onboard Sentinel checks with the ground-based Sentinel for a corresponding input at the operator's control station. If no corresponding input is found, the Sentinel alerts the operator that the system may be under cyber-attack. The alert is delivered by the Cyber Commander station shown in on the right in Figure 9. This is an application that runs on the ground control station and provides alerts to the operator in the event of an attack.

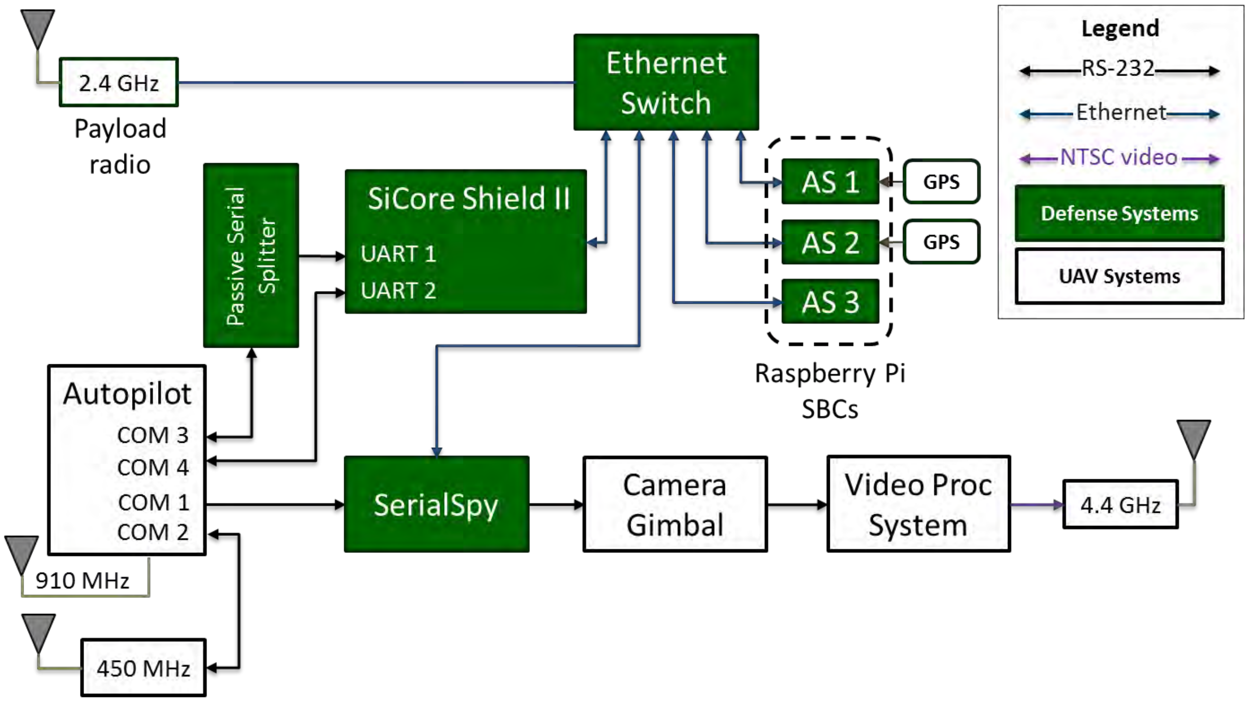


Figure 8. Onboard Sentinel

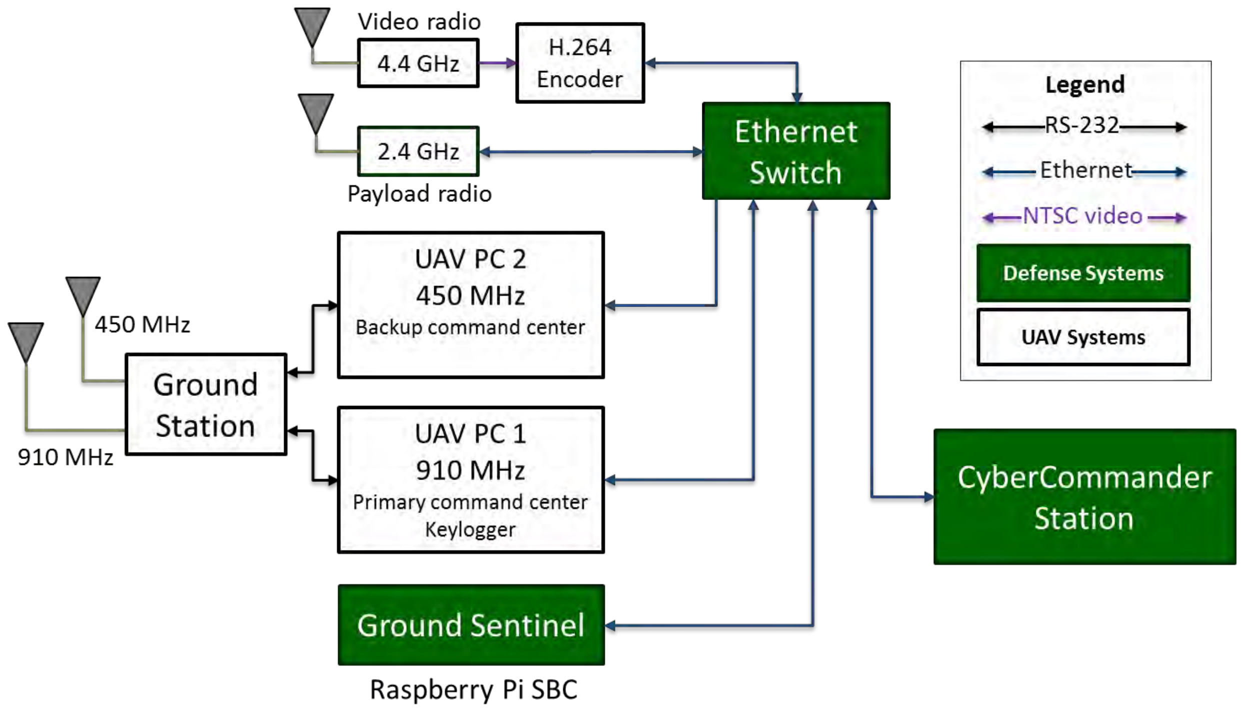


Figure 9. Ground-based Sentinel

6.3 Hardware in the Loop (HIL) Simulation

Figure 10 and Figure 11 show how the components for implementing the cyber-attacks and the Sentinel were integrated with the autopilot system. Three Raspberry Pi single board computers (shown in orange as Attack Pi 1 and Attack Pi 2 in Figure 10 and as Masking Attack Pi in Figure 11) are used to effect the cyber-attacks. Attack Pi 1 executes the waypoint attack onboard the aircraft which changes the list of waypoints in the autopilot's flight plan. This attack causes the aircraft to deviate from its intended course. The Masking Attack Pi at the ground control station hides the fact that the aircraft has gone off course. It does this by intercepting the aircraft's position from the telemetry stream and modifying the coordinates so that the aircraft appears to be on the desired route.

The second onboard attack Pi intercepts and corrupts the GPS position data being sent from the autopilot to the camera gimbal. This attack corrupts the target geolocation information that is included as metadata in the video stream. The third attack originates at the ground control station and affects the gimbal commands (the 'malfunction' attack). This attack software is hosted on the Test Director's Station which also hosts the Tester's Interface software. The Test Director's Station allows the test director to set up attacks and trigger attacks.

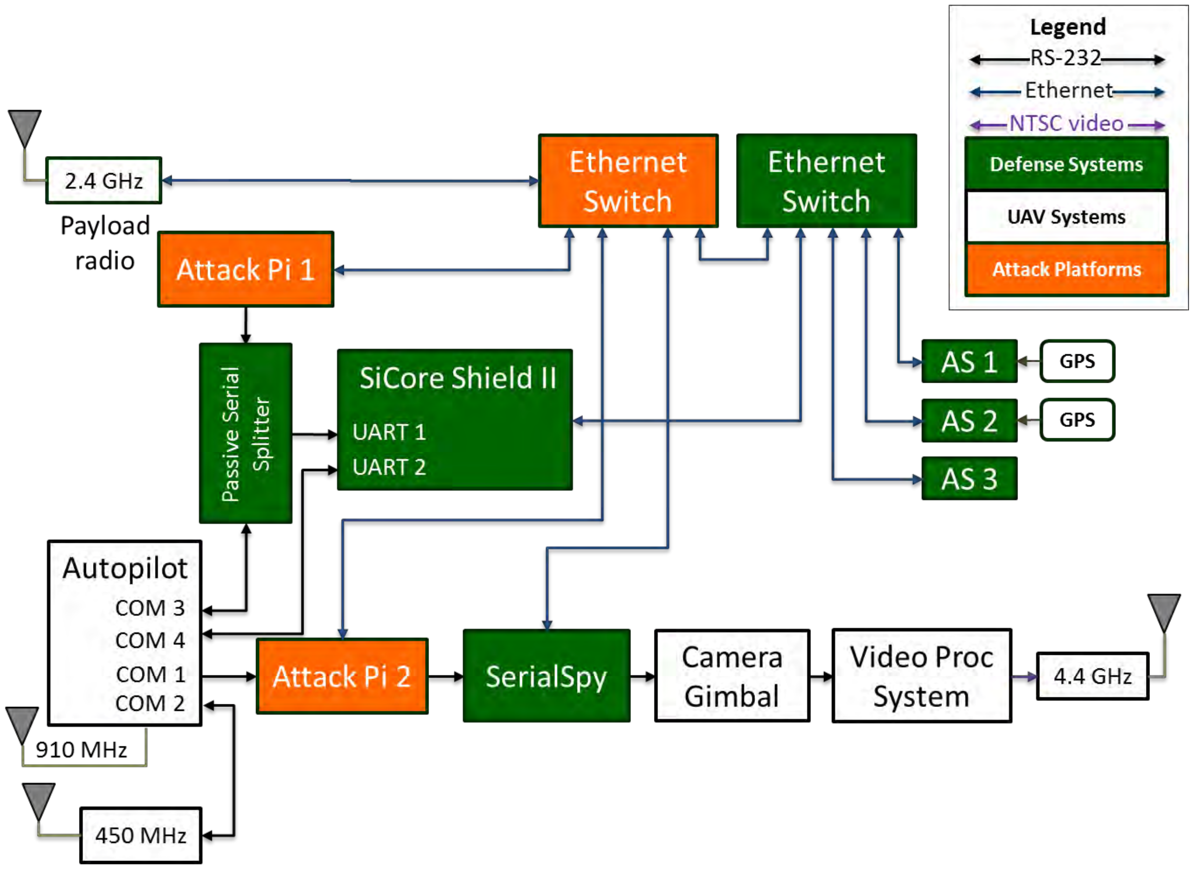


Figure 10. Airborne System Architecture

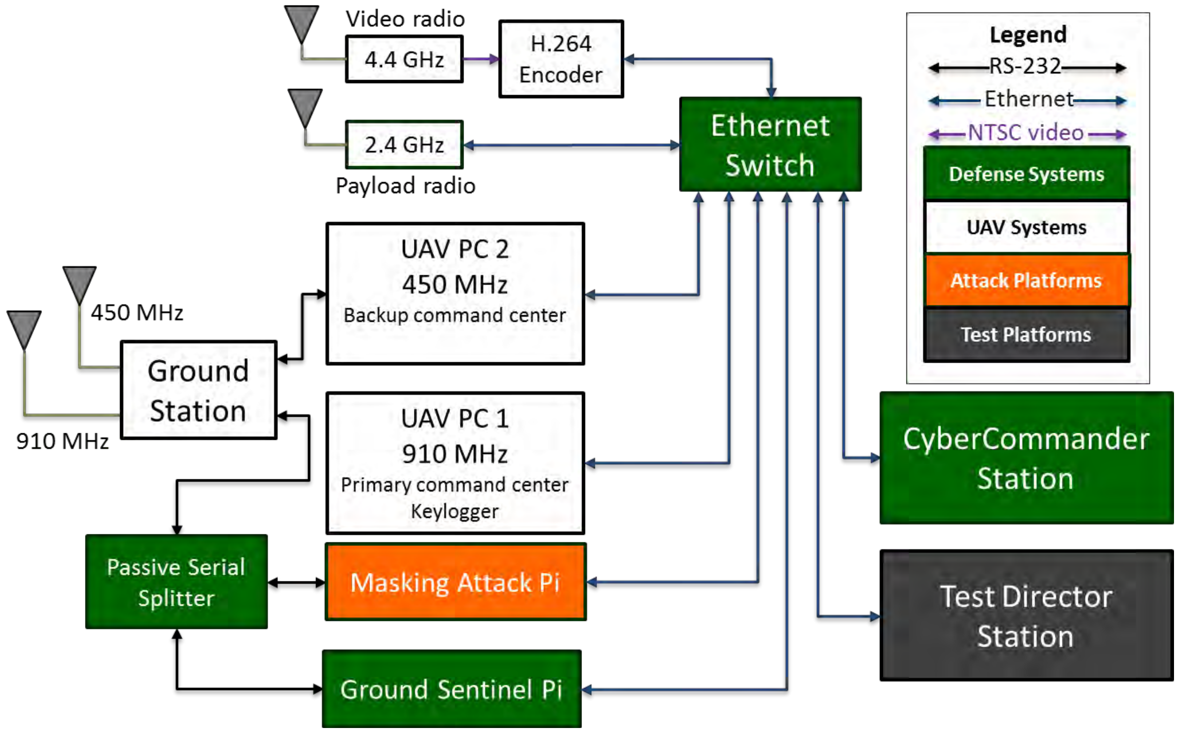


Figure 11. Ground System Architecture

The Piccolo autopilot system includes a hardware-in-the-loop (HIL) and software-in-the-loop (SIL) simulation capability that is essential for properly setting autopilot parameters and tuning gains. While simulation cannot replace flight testing, it greatly reduces the risk by allowing software bugs and other problems to be found in the lab. Figure 12 shows the baseline HIL simulator set-up for GAUSS. The heart of the simulator is the six degree of freedom (6 DOF) flight dynamics model running on one of the lab's computers and the CAN interface to the Piccolo autopilot. The Piccolo sends servo control commands over the CAN bus to the 6 DOF simulation running on the computer. The simulation calculates the dynamic response of the aircraft and sends aircraft state data (sensor data) over the CAN interface to the autopilot. This same aircraft state data is also provided to the CAN interface for the camera gimbal to simulate the gimbal's IMU sensors. The autopilot communicates to two computers running the operator interface software, Piccolo Command Center (PCC). One link is through the ground station over the 910 MHz radio and the other link is through a serial connection on one of the autopilot COM ports to simulate the 450 MHz link. To simulate the analog video signal from the camera, we use the 3D visual simulator software MetaVR to generate the scene as it would be seen by the camera in flight. The analog video output from the video processing system is converted to Ethernet format with an H.264 encoder and routed to a computer running the ViewPoint software. Figure 13 shows the actual simulator hardware with the cyber-attack hardware and Sentinel components installed.

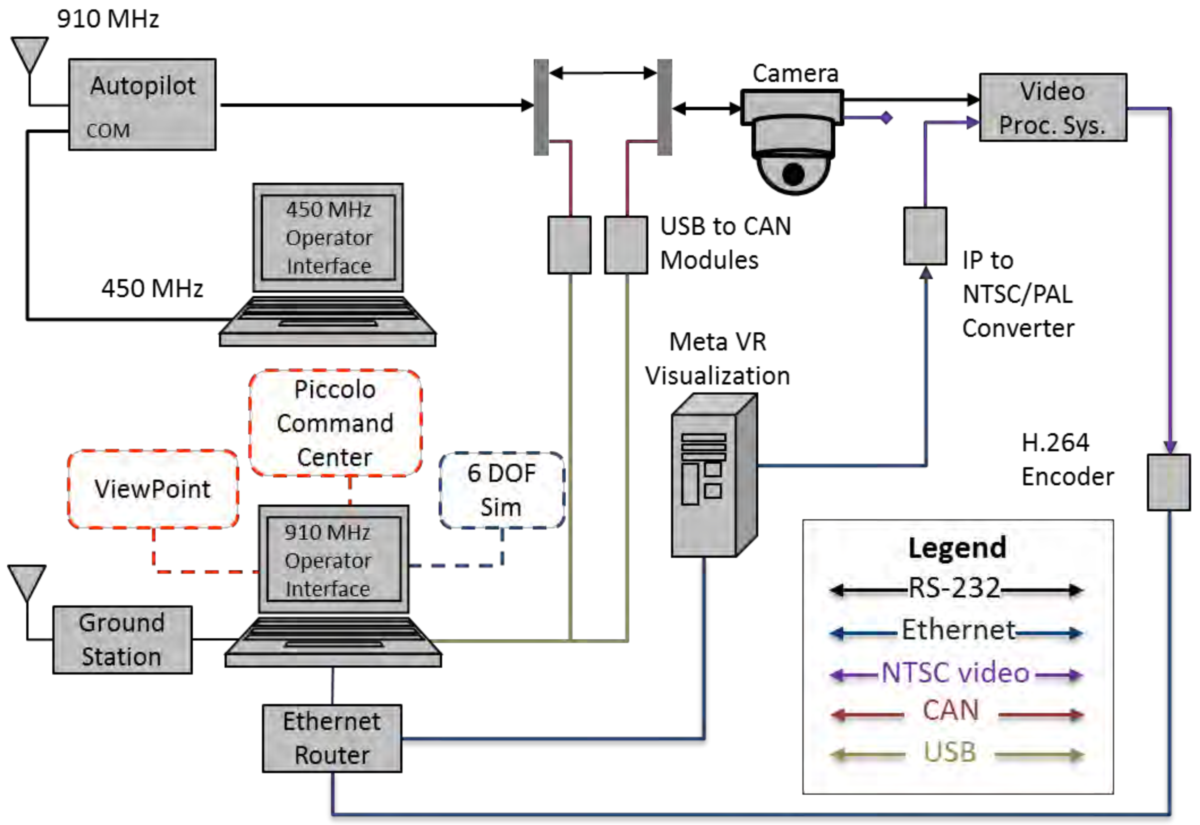


Figure 12. Baseline HIL Set-Up for GAUSS

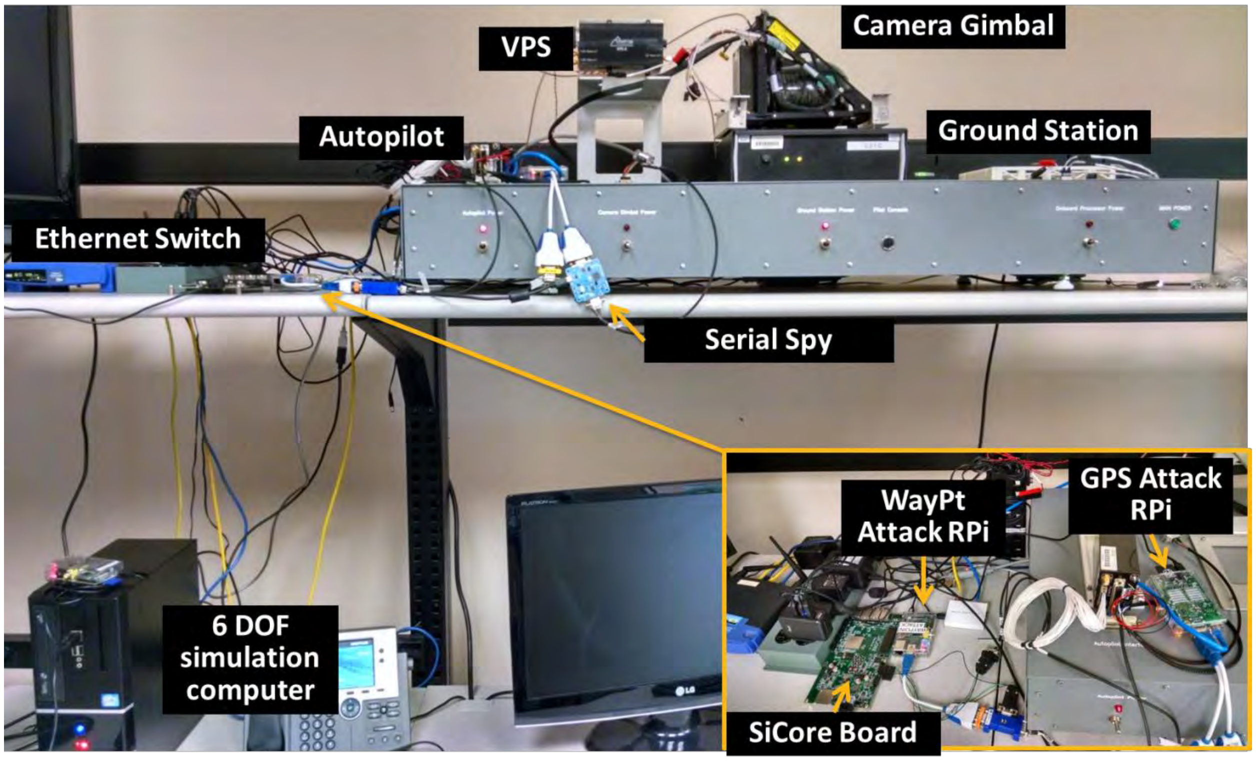


Figure 13. GTRI HIL Simulator with System Aware Cyber Security Components Installed

During the simulation stage of testing we found the HIL simulator to be an indispensable tool for debugging many of the communications issues between the various components on the network. In addition, the simulator allowed us to thoroughly rehearse each attack scenario in preparation for the succeeding flight test. These rehearsals were critical to ensuring that the aircraft's airworthiness was not compromised with the addition of the System Aware attack and defense components.

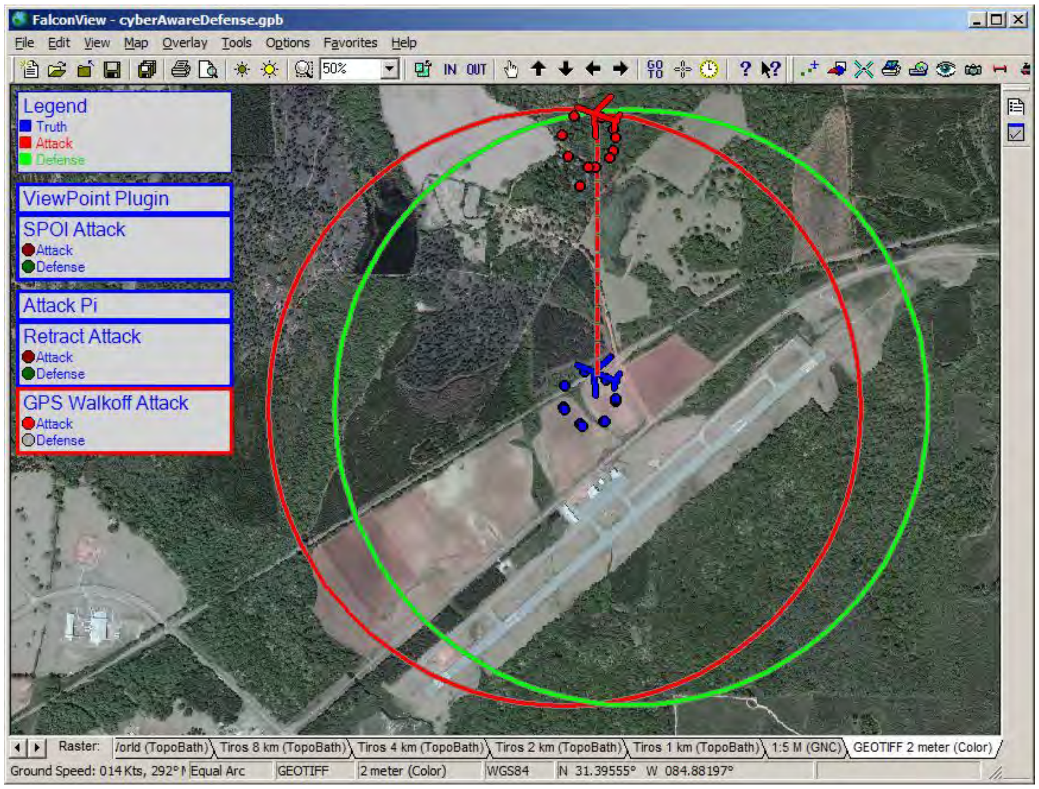
6.4 Tester's Interface

The tester's interface (shown on the right in Figure 11 as the Test Director Station) was developed primarily to allow the test director to monitor the aircraft's true state while it is undergoing a cyber-attack. For example, the waypoint attack takes command of the UAV's flight plan while masking the attack on the operator's ground control station. As a result of the masking, the operator's display shows the aircraft on the intended route while, in reality, the aircraft's flight path is being rerouted. The tester's interface takes advantage of GAUSS' dual radio links for command and control operating at 450 and 910 MHz. Two separate instances of the operator interface software are run at the ground control station, one for each frequency. While the masking attack is underway on the 910 MHz ground control station, the tester's interface uses the 450 MHz control station to monitor the true status of the aircraft.

The situational display is based on GTRI's FalconView map display software. A FalconView plugin visualizes the current state of the system (see Figure 14). As indicated by the legend in the upper left corner of the display, the interface can show where the aircraft really is (Truth, in blue), where the aircraft operator sees it on his Piccolo Command Center (PCC) and where camera operator sees it on ViewPoint while the attack is happening (Attack, in red), and where the defensive systems indicate the aircraft's correct location is (Defense, in green). To do this, the tester's interface takes advantage of GAUSS' dual radio links for command and control operating at 450 and 910 MHz. Two separate instances of the operator interface software are run at the ground control station, one for each frequency. While the masking attack is underway on the 910 MHz ground control station, the tester's interface uses the 450 MHz control station to monitor the true status of the aircraft.

On the display the aircraft icons are in different colors to indicate the aircraft's true position, attack position, and defense position. The aircraft's true position shown in blue indicates where the aircraft really is. This data is retrieved from the 450 MHz Piccolo ground station. The attack position, shown in red, indicates where the operator's station is displaying the aircraft's position during a waypoint attack with masking. It also indicates where camera operator's display is showing the aircraft position in the case of the GPS walk-off attack. The defense position, shown in green, indicates the aircraft location as determined by the Sentinel in response to a waypoint or GPS walk-off attack.

The indicators below the legend give information about the status of various applications. When the FalconView plugin is actively communicating with those components, these boxes expand as shown in Figure 15. The Component Indicators light up for each component when an attack or defense occurs. These components, ViewPoint Plugin, Attack Pi, and Waypoint Masking are described in detail later in this section.


FalconView - cyberAwareDefense.gpb
File Edit View Map Overlay Tools Options Favorites Help
50%
IN OUT
Legend
Truth
Attack
Defense
ViewPoint Plugin
SPOI Attack
Attack
Defense
Attack Pi
Retract Attack
Attack
Defense
GPS Walkoff Attack
Attack
Defense
Raster: /orld (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) GEOTIFF 2 meter (Color)

Ground Speed: 014 Kts, 292° | Equal Arc | GEOTIFF | 2 meter (Color) | WGS84 | N 31.39555° W 084.88197°

Figure 14. Tester's Interface Situational Display

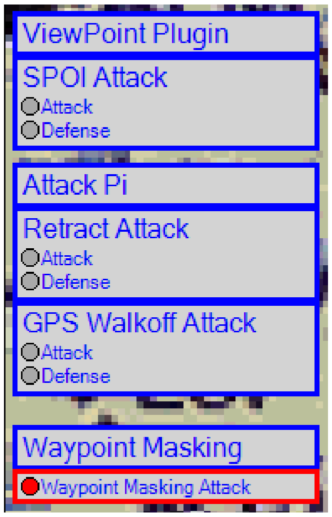
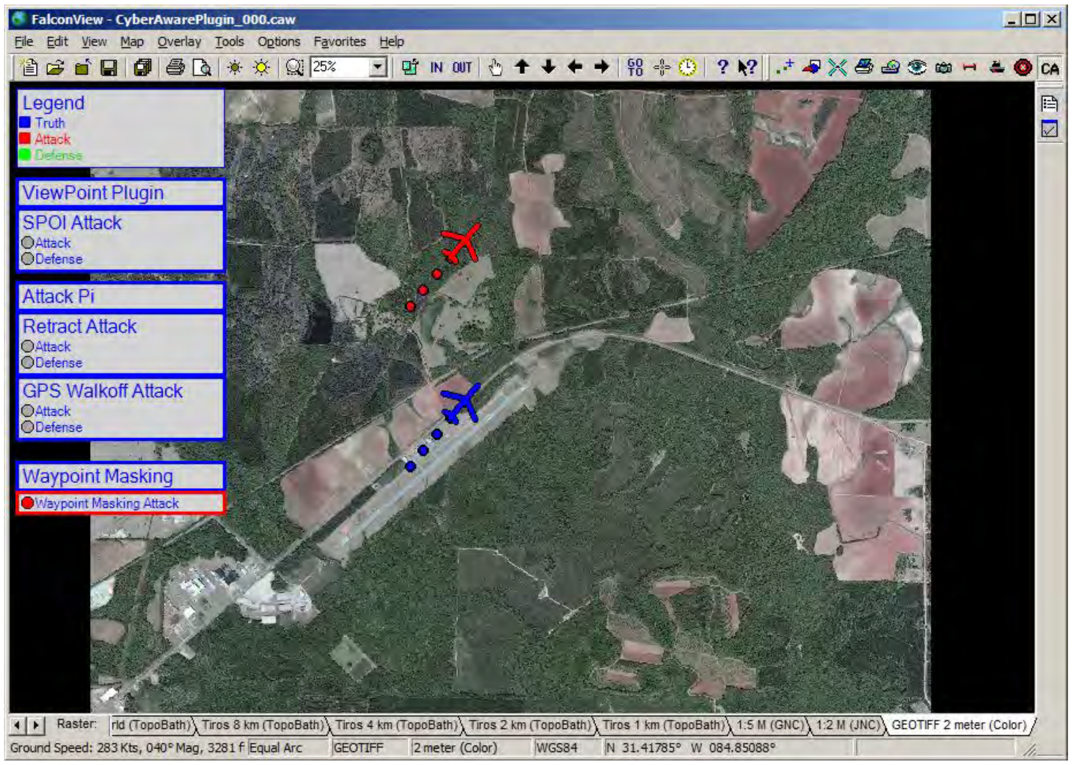


Figure 15. HUD Indicators

Figure 16 shows the Tester's Interface displaying the Aircraft Indicators. These are aircraft icons in different colors that indicate the aircraft's true position, attack position, and defense position. The aircraft's true position shown in blue indicates where the aircraft really is. This data is retrieved from the 450 MHz Piccolo ground station. The attack position, shown in red, indicates where the PCC is displaying the aircraft's position during a waypoint attack with masking. It also indicates where ViewPoint is showing the aircraft position in the case of the GPS walk-off attack. The defense position, shown in green in Figure 17, indicates the aircraft location as determined by the Sentinel in response to a waypoint or GPS walk-off attack.


FalconView - CyberAwarePlugin_000.caw
File Edit View Map Overlay Tools Options Favorites Help
25%
IN OUT
Legend
Truth
Attack
Defense
ViewPoint Plugin
SPOI Attack
Attack
Defense
Attack Pi
Retract Attack
Attack
Defense
GPS Walkoff Attack
Attack
Defense
Waypoint Masking
Waypoint Masking Attack
Raster:
rld (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) 1:2 M (JNC) GEOTIFF 2 meter (Color)

Ground Speed: 283 Kts, 040° Mag, 3281 f Equal Arc GEOTIFF 2 meter (Color) WGS84 N 31.41785° W 084.85088°

Figure 16. Tester's Interface Showing Truth and Attack Positions for the Aircraft

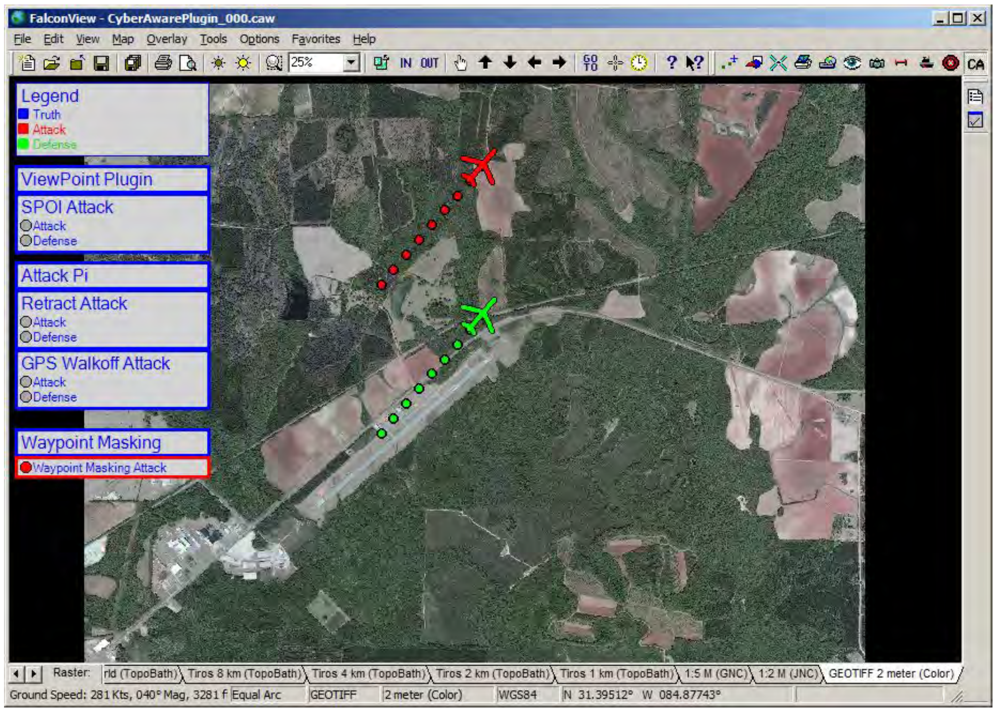
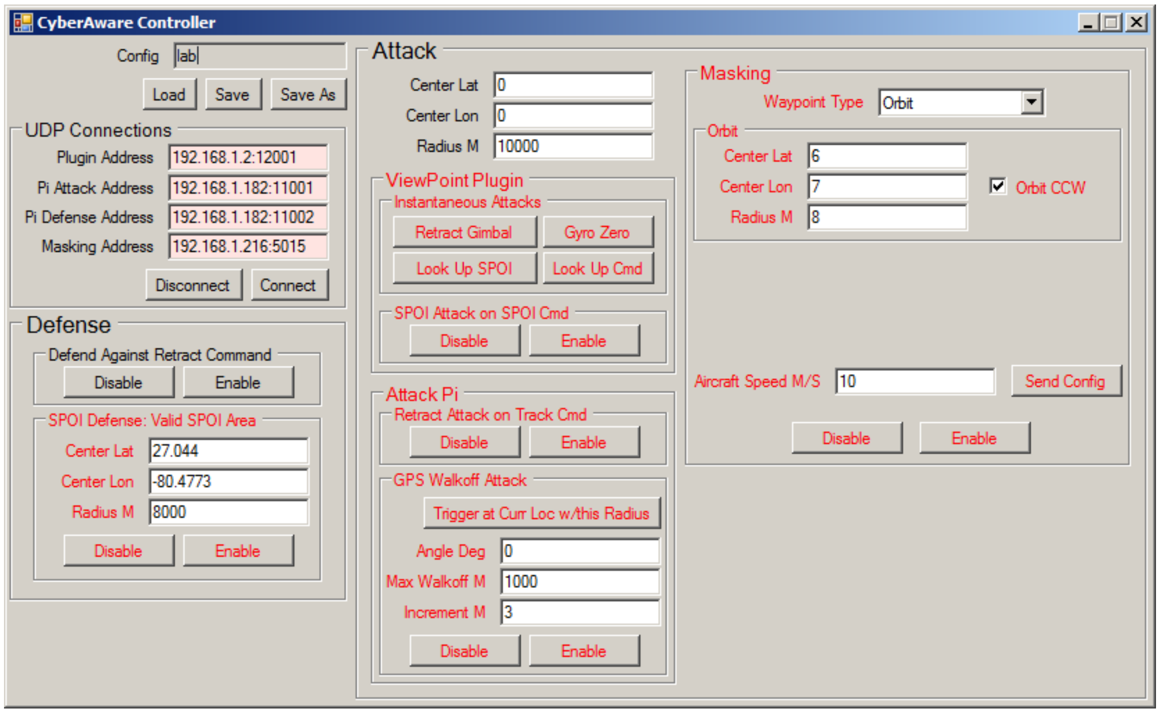

FalconView - CyberAwarePlugin_000.caw
File Edit View Map Overlay Tools Options Favorites Help
Legend
Truth
Attack
Defense
ViewPoint Plugin
SPOI Attack
Attack
Defense
Attack Pi
Retract Attack
Attack
Defense
GPS Walkoff Attack
Attack
Defense
Waypoint Masking
Waypoint Masking Attack
Raster: rld (TopoBath) Tiros 8 km (TopoBath) Tiros 4 km (TopoBath) Tiros 2 km (TopoBath) Tiros 1 km (TopoBath) 1:5 M (GNC) 1:2 M (JNC) GEOTIFF 2 meter (Color)
Ground Speed: 281 Kts, 040° Mag, 3281 f Equal Arc GEOTIFF 2 meter (Color) WGS84 N 31.39512° W 084.87743°

Figure 17. Tester's Interface Showing the Defense Position for the Aircraft (in green)

The tester's interface can also be used to set up several of the cyber-attack scenarios. The main menu for setting the various parameters for an attack is shown in Figure 18. Starting in the upper left corner of the menu, the section labeled "UDP Connections" (see Figure 19) controls the UDP connections to various components in the system. In particular, the "Plugin" refers to the ViewPoint plugin software used to implement the gimbal command attacks. The "Pi Attack" refers to the software on Attack Pi 2 used for the GPS walk-off attack. The "Pi Defense" refers to the software on Attack Pi 2 that could be used to defend against the GPS walk-off attack. However, the defense against the GPS walk-off was also available on the SerialSpy (at address 192.168.1.205) and during testing it was decided that the attack software and the defense software should be hosted on separate devices. This was done to avoid potential confusion over the roles of the various components for attack and defense. The "Masking Address" refers to the Masking Attack Pi at the ground station as shown in Figure 11. When the UDP receive and transmit sockets are successfully created, the text box turns from red to green as shown in Figure 19 (b).


CyberAware Controller

Config: lab

Load | Save | Save As

UDP Connections
- Plugin Address: 192.168.1.2:12001
- Pi Attack Address: 192.168.1.182:11001
- Pi Defense Address: 192.168.1.182:11002
- Masking Address: 192.168.1.216:5015

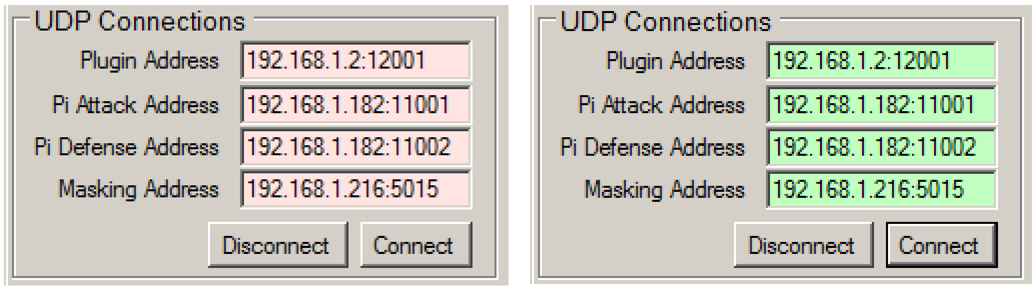
Disconnect | Connect

Defense
- Defend Against Retract Command: Disable | Enable
- SPOI Defense: Valid SPOI Area
  - Center Lat: 27.044
  - Center Lon: -80.4773
  - Radius M: 8000
  - Disable | Enable

Attack
- Center Lat: 0
- Center Lon: 0
- Radius M: 10000
- ViewPoint Plugin
  - Instantaneous Attacks: Retract Gimbal | Gyro Zero | Look Up SPOI | Look Up Cmd
  - SPOI Attack on SPOI Cmd: Disable | Enable
- Attack Pi
  - Retract Attack on Track Cmd: Disable | Enable
  - GPS Walkoff Attack: Trigger at Curr Loc w/this Radius
    - Angle Deg: 0
    - Max Walkoff M: 1000
    - Increment M: 3
    - Disable | Enable

Masking
- Waypoint Type: Orbit
- Orbit
  - Center Lat: 6
  - Center Lon: 7
  - Radius M: 8
  - [✓] Orbit CCW
- Aircraft Speed M/S: 10
- Send Config
- Disable | Enable

Figure 18. Tester's Interface


(a) UDP Connections
- Plugin Address: 192.168.1.2:12001
- Pi Attack Address: 192.168.1.182:11001
- Pi Defense Address: 192.168.1.182:11002
- Masking Address: 192.168.1.216:5015

Disconnect | Connect

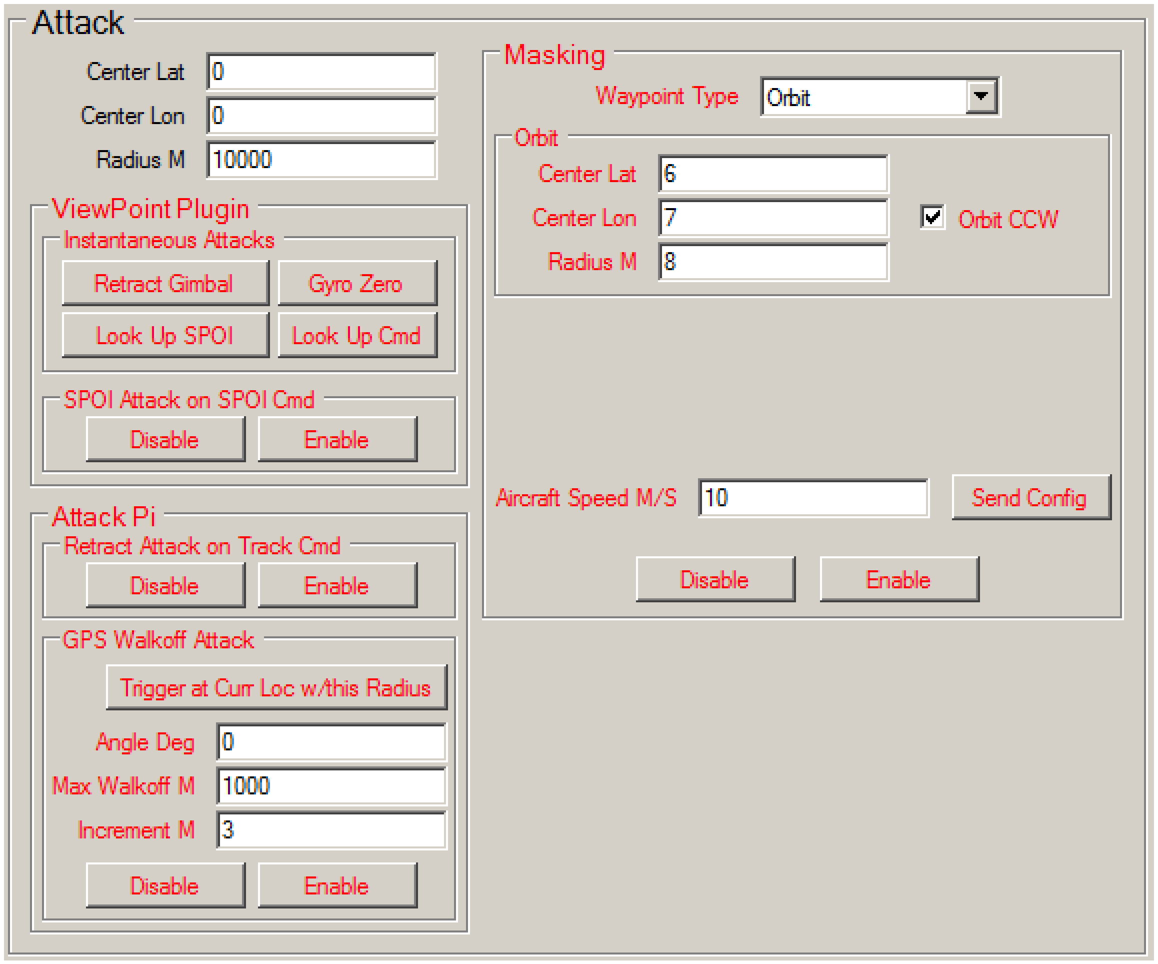
(b) UDP Connections
- Plugin Address: 192.168.1.2:12001
- Pi Attack Address: 192.168.1.182:11001
- Pi Defense Address: 192.168.1.182:11002
- Masking Address: 192.168.1.216:5015

Disconnect | Connect

(a) (b)

Figure 19. UDP Connections Control (a) Sockets not Connected, (b) Sockets Connected

The Attack section (Figure 20) controls with the ViewPoint plugin attack component (gimbal command attacks) and the Attack Pi 2 attack software (GPS walk-off). The "Center Lat" and "Center Lon" parameters allow the tester to set the latitude and longitude of the center of the geographic area in which attacks will occur. The "Radius" is used to specify the radius of the attack region.



Center Lat 0
Center Lon 0
Radius M 10000

Angle Deg 0
Max Walkoff M 1000
Increment M 3

Waypoint Type Orbit
Center Lat 6
Center Lon 7
Radius M 8
[✓ Orbit CCW]
Aircraft Speed M/S 10

Figure 20. Attack Section of the Tester's Interface

The section labeled "ViewPoint Plugin" (see Figure 21) is used to control the gimbal command attacks. By clicking the appropriate button, the tester can initiate several instantaneous attacks on various gimbal commands:

Retract Gimbal – sends a retract gimbal command which causes the gimbal to retract into the aircraft's fuselage

Gyro Zero – sends a Gyro Zero command zeros all the gimbal angles and throws off the calibration of the gimbal

Look Up SPOI – send a command to point the camera directly upward using the sensor point of interest (SPOI) gimbal command message

Look Up Cmd – sends a command to point the camera directly upward using the camera angle command message

The SPOI Attack on SPOI Cmd section allows the tester to enable or disable the SPOI attack. Enabling this attack causes the camera to point upward whenever the payload operator tries to lock onto a spot on the ground using the SPOI command. The attack modifies the SPOI command by setting the point of interest directly above the aircraft's location so that it will point upward instead of pointing at the correct location on the ground.

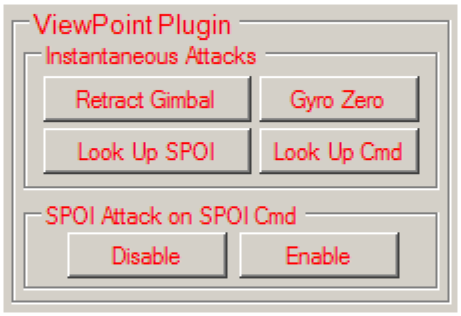
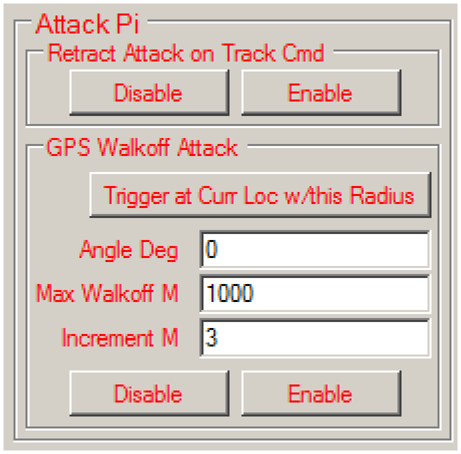


Figure 21. ViewPoint Plugin Section

The “Attack Pi” subsection (Figure 22) is used enable a gimbal retract attack and to set up and enable the GPS walk-off attack. The “Retract Attack on Track Cmd” section allows the tester to enable or disable the gimbal retract attack. When the attack is enabled and the aircraft enters the geographic region for the attack, the camera gimbal retracts whenever the operator tries to engage the tracking feature. The ViewPoint plugin with the attack software (hosted on the Test Director station) sends a retract command to the gimbal when the operator clicks on a spot in the video in order to lock the camera on an object for tracking. As a result, this commonly-used gimbal feature is rendered useless unless the corresponding defense is active on the SerialSpy.

The “GPS Walkoff Attack” subsection controls how the GPS position that is being fed from the autopilot to the camera gimbal is corrupted. The GPS walk-off attack offsets the true GPS location of the aircraft by the specified Increment in the compass direction specified by Angle every time a GPS update occurs. The “Max Walkoff” limits the magnitude of the position error. The corrupted data is then passed on to the camera gimbal and is included in the video metadata that is sent to ViewPoint. Recall that the location and radius in which this attack will be triggered is set at the top of the Attack section (see Figure 20).

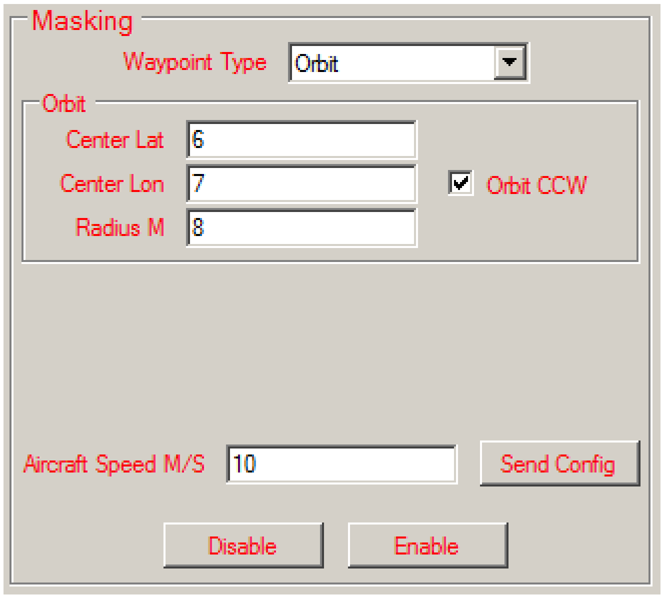
The “Trigger at Curr Loc w/this Radius” allows the tester to initiate the GPS walk-off attack at the aircraft’s current location rather than waiting until the aircraft enters the attack region. It does so by setting the center of the attack region to the aircraft’s current location.


Attack Pi
Retract Attack on Track Cmd
Disable
Enable
GPS Walkoff Attack
Trigger at Curr Loc w/this Radius
Disable
Enable

Angle Deg: 0
Max Walkoff M: 1000
Increment M: 3

Figure 22. Attack Pi Section

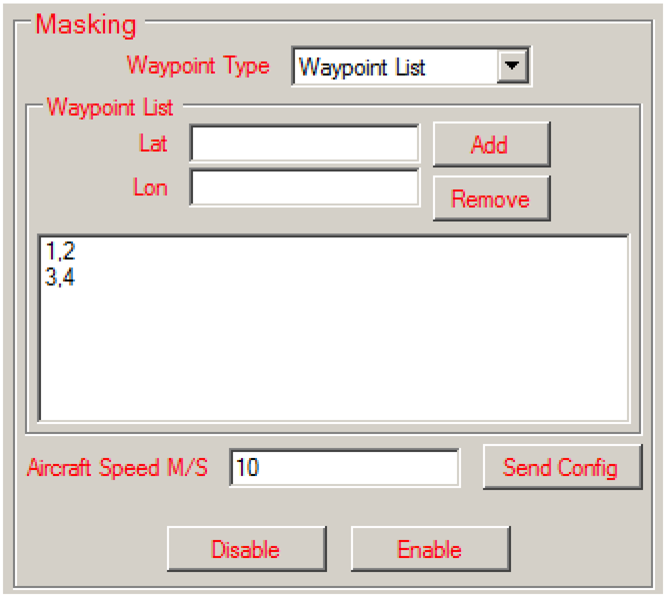
The "Masking" section of the Tester's Interface, shown in Figure 23, allows the tester to set up and enable the waypoint attack with masking. This attack hijacks the aircraft while hiding the attack from the aircraft operator on the Piccolo Command Center.



Waypoint Type: Orbit
Center Lat: 6
Center Lon: 7
Radius M: 8
Orbit CCW: ✓
Aircraft Speed M/S: 10

Figure 23. Masking Section of Tester's Interface (Orbit Waypoint Selection)

The "Waypoint Type" drop-down menu lets the tester specify whether the waypoints for hijacking the route are either an orbit waypoint or a waypoint list (for a route). If an orbit waypoint is specified, the tester can set the latitude and longitude of the center of the orbit and the radius of the orbit. If the tester wants to send a list of false waypoints, he can create a list by adding waypoints and specifying their latitude and longitude (Figure 24). The "Aircraft Speed" box allows the tester to set the speed of the aircraft once the attack has taken control of the aircraft.



Waypoint Type: Waypoint List

1,2
3,4

Aircraft Speed M/S: 10

Figure 24. Masking Section of Tester's Interface (Waypoint List Selection)

6.5 Aircraft Integration

6.5.1 System Aware Payload

Integration of the Sentinel components and cyber-attack devices into the GAUSS air vehicle required the design and fabrication of wiring harnesses, additional GPS antenna mounts, and a chassis to hold all of the components (Figure 25). This chassis was installed in the aircraft's forward payload bay as indicated in Figure 26.

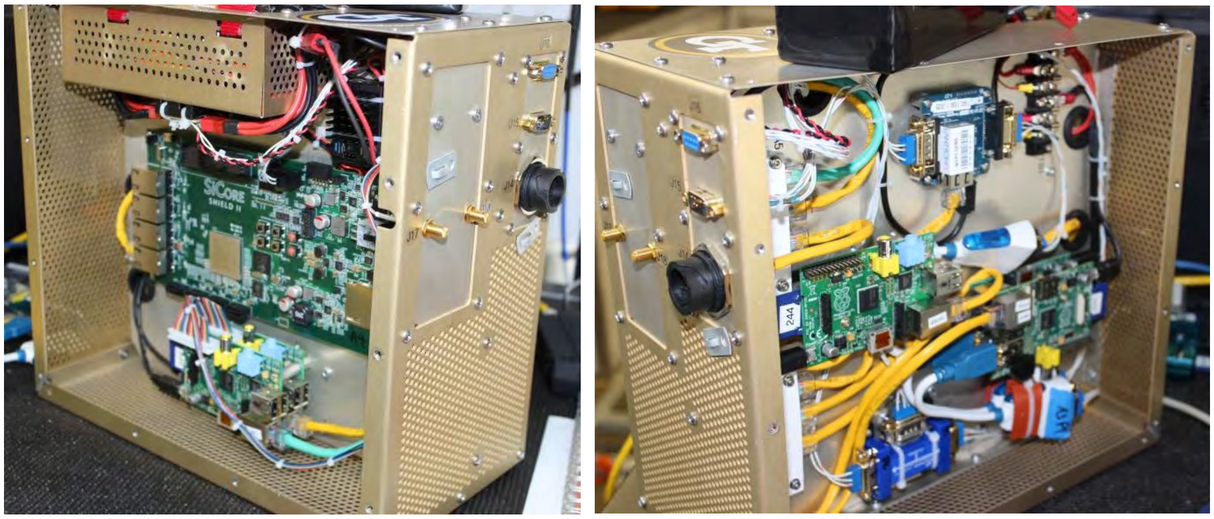


Figure 25. Payload Chassis

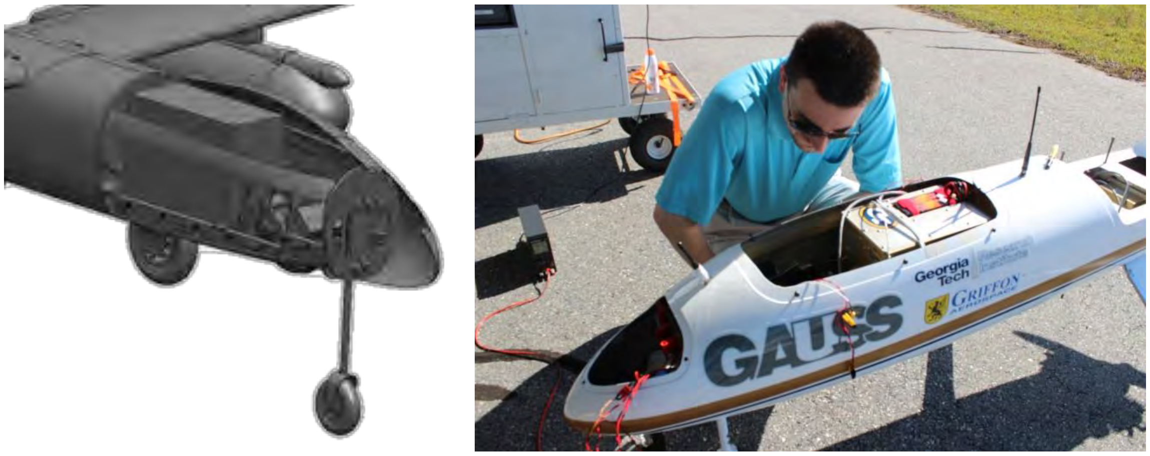


Figure 26. Payload installation in GAUSS

6.5.2 Airworthiness Qualification

Integrating systems such the System Aware Sentinel and the cyber-attack components into a UAS requires extra attention to safety and airworthiness. GTRI follows an airworthiness qualification (AQ) process similar to that outlined for UASs in (MIL-HDBK-516B). The primary focus of the AQ process for this demonstration was ensuring that the System Aware test components (the Sentinel and cyber-attack systems) did not compromise the ability of the autopilot system, including data links, to function properly at all times.

The goal of the AQ process is to ensure that the UAS meets the FAA's primary concerns for operations in the national airspace system (NAS):

- Containment – the ability of the aircraft to be contained within the proposed flight area
- Lost link – the sequence actions that the UAS will execute in the event the command and control link is lost leading to a safe resolution
- Flight conclusion – an independent means to safely conclude a flight in the event the command and control cannot be reestablished

Many desired safety features are standard functions on today's autopilots such as the Piccolo II. These include programmable behaviors for what the aircraft should do in the event communications and GPS are lost. In the case of lost communications, if the autopilot has not heard from the ground station within a specified amount of time (e.g., 20 seconds), it can be programmed to have the aircraft orbit in a predetermined flight plan, typically near the ground station. If communications are not reestablished before the flight timer expires, the autopilot can be set to execute an auto-landing at a predefined waypoint. If GPS is lost, the autopilot can rely on the inertial navigation system (INS) alone for a short period of time. After the GPS timeout expires, the autopilot will assert a flight termination command which can be used to trigger several actions such as closing the throttle and commanding maximum right aileron and rudder and up elevator for an aerodynamic termination. For aircraft equipped with a flight recovery system, such as a parachute, the flight termination signal can be used to deploy the parachute.

To help ensure containment, the Piccolo autopilot can be programmed with a three dimensional airspace boundary called a geo-fence. The geo-fence coordinates reside in the autopilot and they set hard limits on where the UAS can be commanded to go. In addition to preventing the creation or execution of flight plans that would take the aircraft outside of the fence, flight termination can be asserted if the aircraft flies outside the fence. The geo-fencing feature was not used during the demonstration program because the aircraft was kept within a fairly tight pattern around the airport under the watchful eye of the safety pilot.

In addition to the standard safety features of the Piccolo autopilot system, GTRI's GAUSS employs several additional safety measures. These include a secondary radio link at 450 MHz for command and control. As mentioned previously, this link is used to provide truth data on the aircraft's location when the 910 MHz link is under cyber-attack. The aircraft is also equipped with two orthogonal 2.4 GHz receivers as part of the wireless pilot manual control system. Thus, the aircraft can be controlled manually via the 910 MHz link using the wired pilot console connected to the ground station or through the 2.4 GHz link using the wireless pilot console.

Other safety features include a kill switch on the power supply for the payload, an emergency retrieval beacon, and a three axis magnetometer. Because of the potential for the payload system to interfere with the aircraft's operation, a kill switch was added to the payload power supply circuit to enable the flight test engineer to kill power to all of the payload components via a discrete command from the autopilot system. This was done to ensure that all of the non-flight critical components communicating with the autopilot could be shut down in the event of an emergency. A schematic diagram of the kill switch circuit is provided in Appendix C.

All fixed-wing unmanned aircraft flown by GTRI carry an emergency retrieval beacon. This device transmits a periodic tone at about 219 MHz which can be used to locate a downed aircraft. This commercially available product can operate for up to 2 weeks using a primary lithium battery. Work is under way to integrate this capability directly into the on board avionics. The goal of this work is to provide more reliable operation and reduced cost.

All unmanned aircraft typically navigate using GPS. In the event of a loss of GPS, the aircraft can fly away or, if so configured, immediately ditch. Both options are undesirable. When resources permit, GTRI equips its UAVs with an electronic compass referred to as a magnetometer. This allows the aircraft to approximate its heading and navigate via "dead reckoning". While this is not as accurate as GPS navigation, it can provide some time for GPS recovery or a more controlled flight termination. GTRI's GAUSS is equipped with a Honeywell HMR-2300 3-axis magnetometer which feeds into the navigation filter; although, as noted earlier, the magnetometer was not used during this demonstration. Work is underway to produce a smaller, more integrated magnetometer to more easily integrate this capability in test aircraft. The current design is smaller, lighter, and less expensive than the Honeywell HMR-2300. This is possible using very small magnetometers found in devices like cell phones.

6.6 Flight Testing

6.6.1 Flight Conditions and Restrictions

The flight test demonstration was conducted at the Early County airport in Blakely, GA under a Certificate of Authorization (COA) from the FAA. The COA permits operations in Class E and Class G airspace at or below 5000 ft above ground level in the vicinity of the airport (see Appendix D). Other operational restrictions include:

- Operations must be in visual meteorological conditions during daylight hours
- No operations (including lost link procedures) over populated areas
- Sterile cockpit procedures during all critical phases of flight
- All crew members must have a current second class airman medical certificate and current crew resource management training
- The pilot in command must hold, at a minimum, a current FAA private pilot certificate
- Supplemental pilots must have, at a minimum, successfully completed private pilot ground school and passed the written test
- A Notice to Airmen must be issued not more than 72 hours in advance, but not less than 48 hours prior to the operation

As part of the COA application, GTRI submitted a contingency plan that addressed emergency recovery or flight termination of the unmanned aircraft (UA) in the event of unrecoverable system failure. In particular, the plan covered lost link and flight termination procedures based on the autopilot's safety features described in section 6.5.2.

6.6.2 The Test Team

The test team consisted of eight personnel with the following roles and responsibilities:

- Lead flight test engineer – supervised execution of the flight test plan, monitored the aircraft status during the tests, made Unicom radio calls to alert local traffic

- Pilot in command – Supervised flight operations, watched for other aircraft in the vicinity, monitored Unicom radio traffic
- Safety pilot – performed manual takeoffs and landings, watched for other aircraft in the vicinity
- Visual observer – watched for other aircraft in the vicinity
- Autopilot operator – commanded and monitored the aircraft through the autopilot system, operated the camera gimbal system
- System Aware operator 1 – initiated gimbal attacks and GPS walk-off attacks, monitored Sentinel defense responses
- System Aware operator 2 – initiated waypoint and masking attacks, monitored Sentinel defense responses
- Videographer – recorded video and still images to document the demonstration

Figure 27 shows the test team inside the GTRI UAV trailer. The first set of wall-mounted monitors, on the far left in the picture, served as displays for the 910 MHz PCC operator's station manned by the autopilot operator. The second set of monitors was used for the 450 MHz PCC station manned by the flight test engineer. To the right of these monitors was a laptop computer running a Hyperterminal interface to the SerialSpy which implemented the defenses against gimbal command attacks and attacks on the GPS data provided to the gimbal. In the middle of the picture is a laptop computer running the Tester's Interface software. The third set of wall-mounted monitors served as multi-purpose displays. The top monitor displayed the ViewPoint interface during gimbal testing and the Cyber Commander interface during waypoint attack testing. The lower monitor was used to display the status of the various Raspberry Pi SBCs and the SiCore board. Also at this station, was a laptop computer running the interface for the waypoint attack software.

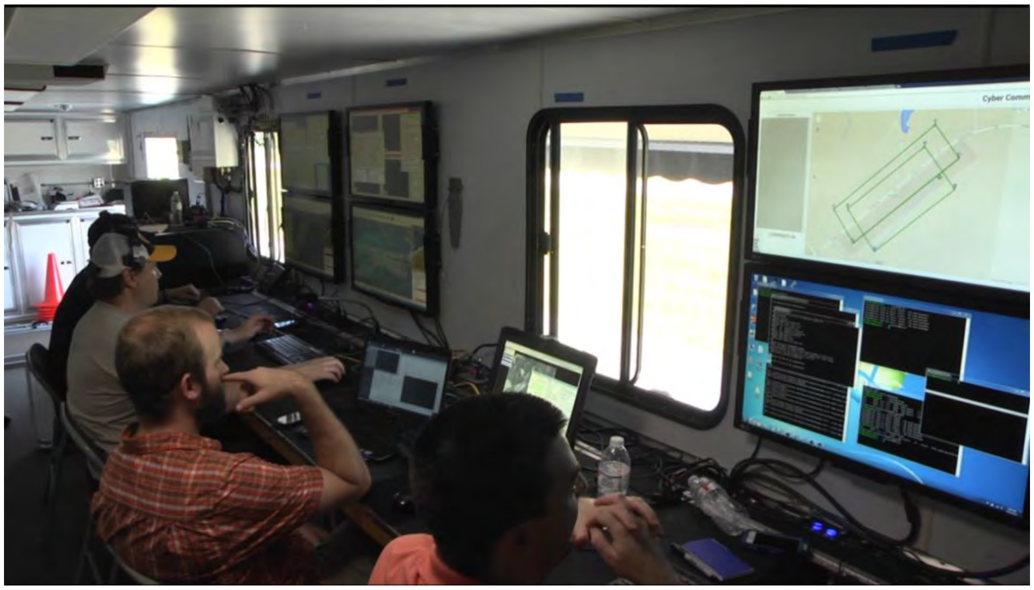


Figure 27. Test Team inside the GTRI UAV Trailer

The test team maintained communications with each member via a Clear-Com radio intercom system. This communication system also allowed the pilot in command and the flight test engineer to talk directly with local traffic over Unicom frequency 122.9 MHz.

6.6.3 Attack Demonstrations

Three types of attacks were demonstrated in flight: a GPS walk-off, a gimbal command attack, and a waypoint attack. Testing began with the GPS walk-off attack because it was the most benign in terms of the potential effects on the aircraft. Next were the gimbal command attacks which would not affect the aircraft's ability to fly, but did pose a slight risk of damage to the camera gimbal. The waypoint attack was scheduled for last because it posed the greatest risk to loss of control over the aircraft. The following paragraphs describe these attacks and the results.

GPS Walk-off: The aircraft's commanded flight path was a right hand pattern on Runway 23 as shown in Figure 28. The geographic region for the attack was defined as a circular area centered to the northeast of the aircraft's flight path with a radius of 609 m. This resulted in a partial overlap with the aircraft's commanded path as shown in Figure 28. As the aircraft crossed into the attack region the GPS walk-off attack was initiated. In this demonstration, the attack was configured to add a growing bias the aircraft's latitude coordinate causing its reported position to shift northward as illustrated in Figure 29. During the first pass through the attack region the Sentinel defense was disabled and the position data being fed from the autopilot to the gimbal was observed to deviate from the true position. During the next pass through the attack zone the defense was enabled and the GPS walk-off was detected almost immediately. The SerialSpy then replaced the corrupted GPS position data with validated data from the Sentinel and forwarded the data to the camera gimbal.

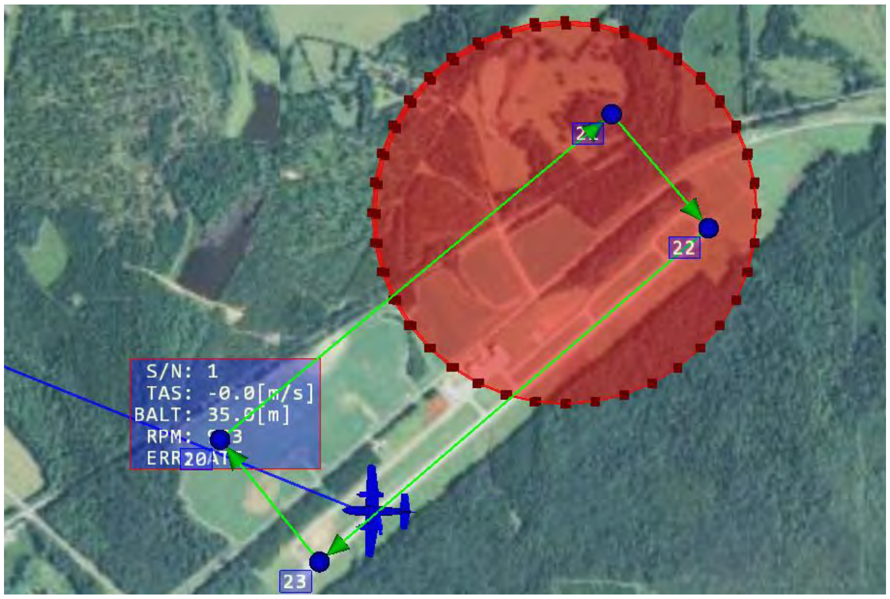


Figure 28. GPS Walk-off Attack Region

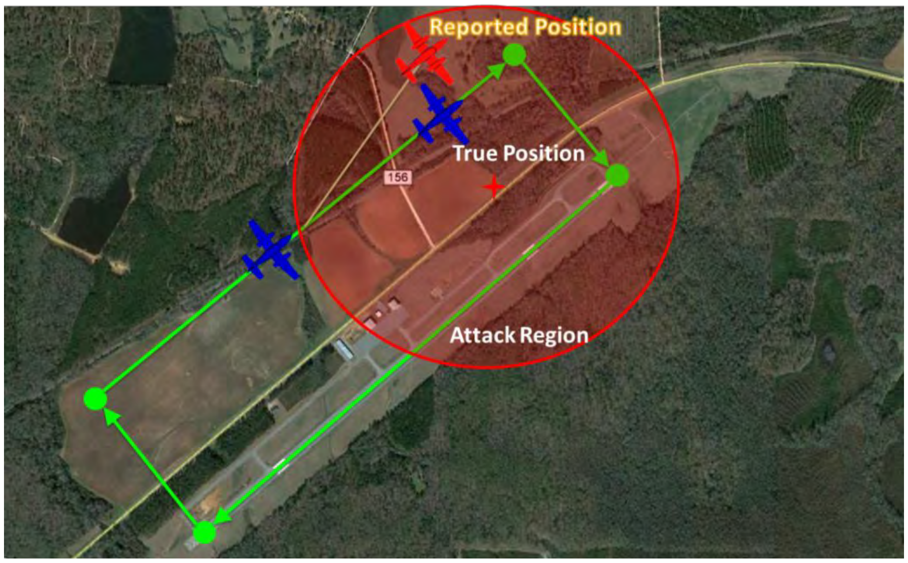


Figure 29. GPS Walk-off Attack Position Report

Gimbal Command Attacks: The retract attack is referred to as Attack 2a and the SPOI attack is referred to as Attack 2b. The aircraft's commanded flight orbit was centered over the runway as shown in Figure 30. The geographic region for the attack was defined as a circular area centered to the northeast with a radius of 609 m. This resulted in a partial overlap with the aircraft's commanded orbit as shown in Figure 30. As the aircraft crossed into the region encompassed by the attack orbit the SPOI attack was initiated (Attack 2b). During the initial pass through the attack region the defense was disabled and the SPOI slewed upwards at the test director's command. During the next pass, the defense was enabled. The defense blocked the SPOI command preventing the camera from being slewed upwards.

After the SPOI attack was demonstrated it was deactivated and the gimbal retract attack (Attack 2a) was enabled. During the initial pass through the attack region the defense was disabled and the gimbal retracted when the operator tried to track an object at the center of the orbit. During the next pass, the defense was enabled. The defense blocked the gimbal retract command when the operator initiated a tracking command and the operator was able to successfully track a ground object.

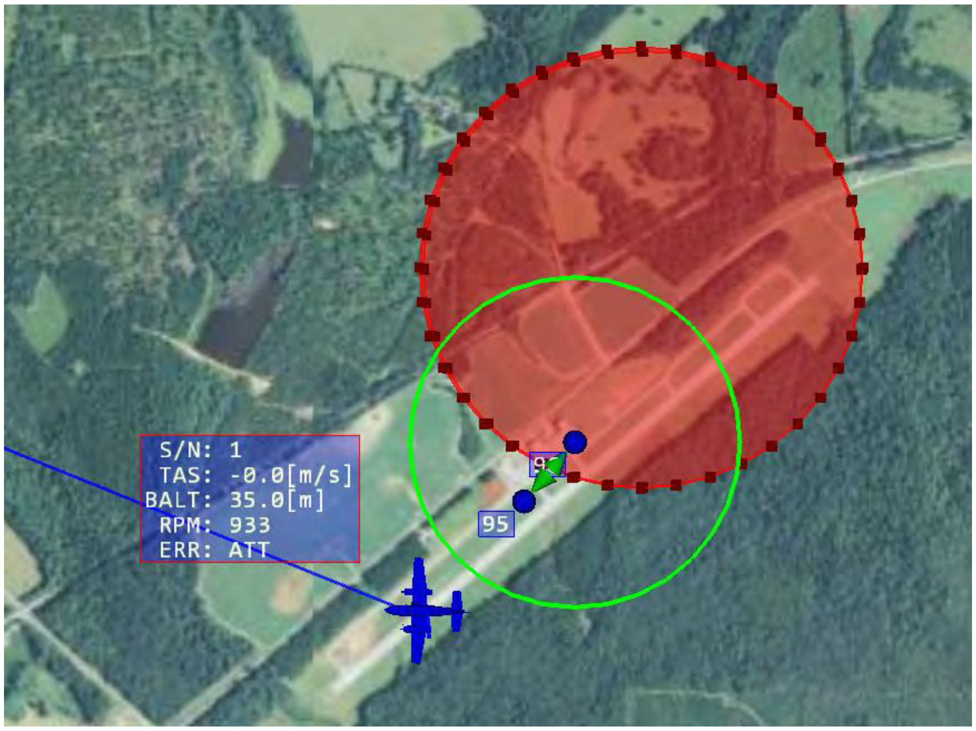


Figure 30. Gimbal Command Attack Region

Waypoint Attack: The waypoint attack changes the waypoints in the autopilot's flight plan causing it to fly a different trajectory from the one intended by the operator. The aircraft's commanded flight plan was one of two rectangular patterns aligned with the runway as shown in Figure 31. The other rectangular pattern was used as the set of waypoints for the attack flight plan. The coordinates of the waypoints are provided in Table 2 and Table 3.

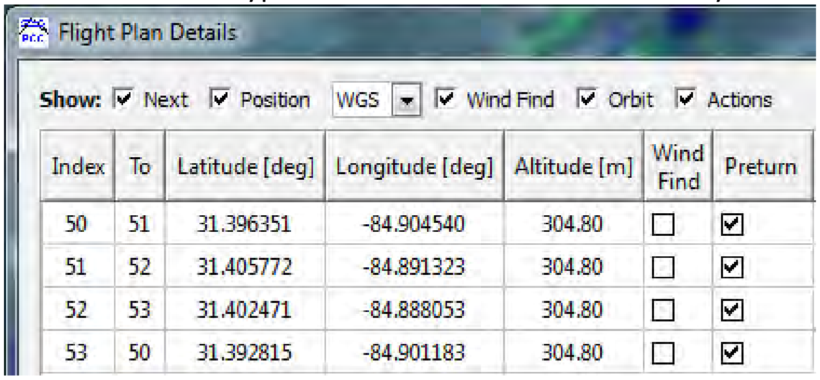
For this demonstration the operator commanded the aircraft to fly the rectangular pattern consisting of the waypoints in Table 2. During the first pass through the pattern the test director initiated a waypoint attack with the defense disabled. The aircraft diverted to the rectangular pattern made up of the waypoints in Table 3. This exercise verified that the attack was working properly with regard to taking control of the aircraft.

The defense was then enabled and the operator commanded the aircraft to return to the original pattern. In this case the Sentinel successfully noted the change in the flight plan, verified that the change came from the operator, and indicated on the Cyber Commander interface that a normal change had occurred. During the next pass through the pattern when the attack was initiated the Sentinel successfully noted the change and, because it could not find an associated operator input for the change, alerted the operator via the Cyber Commander interface of a possible cyber-attack. The Cyber Commander interface offered the appropriate corrective actions to the operator which included restoring the original flight plan, flying the aircraft back to its home location, and flying to another waypoint or flight plan of the operator's choice. For this demonstration the operator elected to have the Sentinel change back to the original plan.

In the next pass through the pattern the masking attack on the 910 MHz ground control station was triggered along with the waypoint attack onboard the aircraft. Unfortunately, the masking portion of the attack did not work properly. When the masking attack was engaged, the aircraft's position froze on the 910 MHz ground control station display. The aircraft's true location could still be seen on the 450 MHz ground control station display.

The failure of the masking attack led to an interesting and unplanned situation. The 910 MHz control station is the primary display used by the operator for controlling the aircraft. When the masking attack froze this station the operator had to switch to the 450 MHz ground station to issue waypoint commands to return the aircraft to the desired pattern. During the setup of the tests, the Sentinel was not made aware of the presence of the 450 MHz ground control. As a result, when the operator initiated the change from this "unknown" station, the Sentinel detected the change as an unverified change and alerted the Cyber Commander station that a cyber-attack was underway. This attack is very similar to a spoofing attack where a false operator station would attempt to take over flight operations for a UAV. This was a critical point in the tests where the Sentinel worked as designed to protect the flight plan in an unplanned scenario and was a defining moment for the tests. The Sentinel was doing what it was designed to do, protect the system functions and offer corrective actions to the operator.

Table 2. Waypoints on northwest side of runway

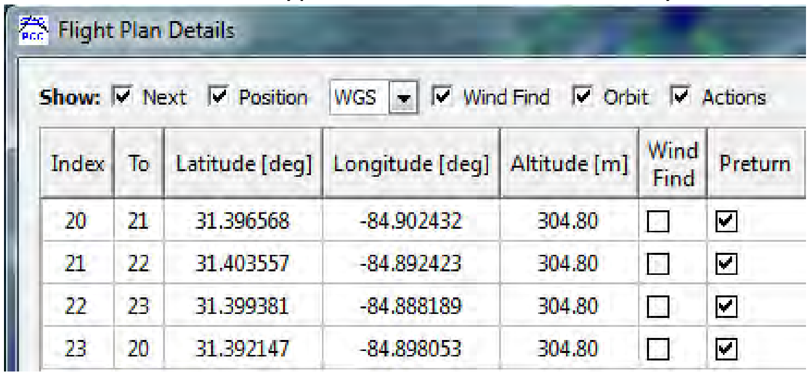


Flight Plan Details

Show: ✓ Next ✓ Position WGS ✓ Wind Find ✓ Orbit ✓ Actions

Index	To	Latitude [deg]	Longitude [deg]	Altitude [m]	Wind Find	Preturn
50	51	31.396351	-84.904540	304.80	☐	✓
51	52	31.405772	-84.891323	304.80	☐	✓
52	53	31.402471	-84.888053	304.80	☐	✓
53	50	31.392815	-84.901183	304.80	☐	✓

Table 3. Waypoints centered over runway



Flight Plan Details

Show: ✓ Next ✓ Position WGS ✓ Wind Find ✓ Orbit ✓ Actions

Index	To	Latitude [deg]	Longitude [deg]	Altitude [m]	Wind Find	Preturn
20	21	31.396568	-84.902432	304.80	☐	✓
21	22	31.403557	-84.892423	304.80	☐	✓
22	23	31.399381	-84.888189	304.80	☐	✓
23	20	31.392147	-84.898053	304.80	☐	✓

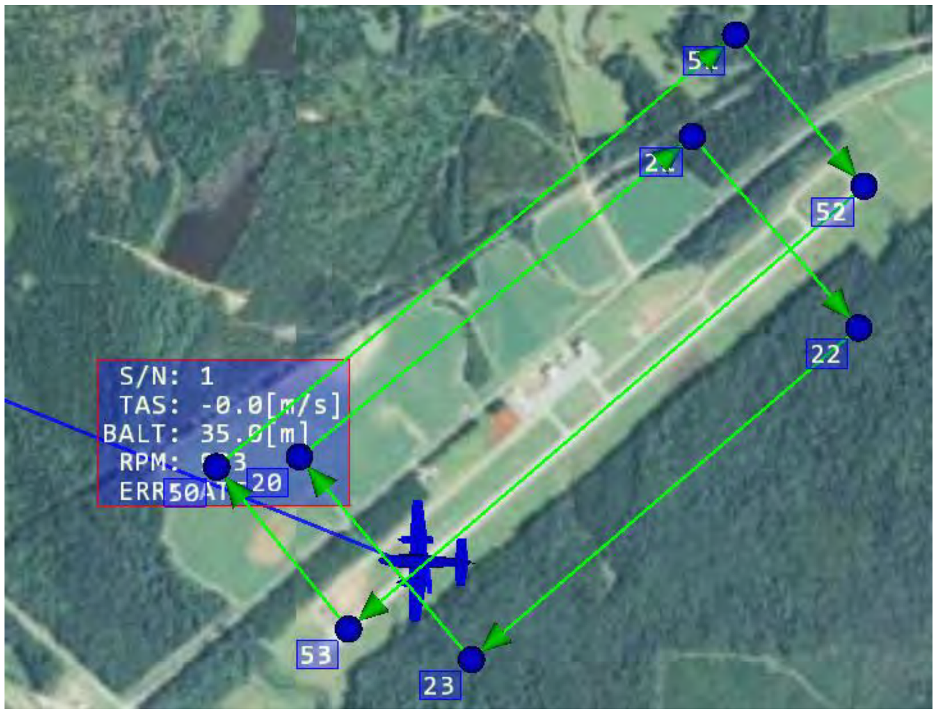


Figure 31. Waypoint Attack Flight Plans

7 References

Gorman, S., Dreazen, Y., & Cole, A. (2009). Insurgents Hack U.S. Drones. *Wall Street Journal, Dec. 17*.

Hartmann, K., & Steup, C. (2013). The Vulnerability of UAVs to Cyber-Attacks - An Approach to Risk Assessment. *5th International Conference on Cyber Conflict, NATO CCCD COE Publications*.

Jones, R. A., & Horowitz, B. M. (2012). A System-Aware Cyber Security Architecture. *Systems Engineering, Vol. 15, No. 2*.

Kerns, A., Shepard, D. P., Bhatti, J. A., & Humphreys, T. E. (2014). Unmanned Aircraft Capture and Control via GPS Spoofing. *Journal of Field Robotics, Vol. 31*, 617-636.

MIL-HDBK-516B. (2005). Airworthiness Certification Criteria. *Department of Defense Handbook*.

Appendix A: SerialSpy

The SerialSpy is a microcontroller-based device designed to monitor and/or manipulate RS-232 serial data between two devices. It uses a Microchip dsPIC33EP digital signal controller to passively monitor bidirectional serial data transparently from the host serial devices, insert new serial data packets, modify serial data packets, or completely block specific serial data. The SerialSpy is controlled through a USB serial interface or through telnet over Ethernet.

The SerialSpy is designed to be inserted between two devices connected via RS-232. In the UAV implementation of the Sentinel, the SerialSpy is inserted between the Piccolo autopilot and the camera gimbal as shown in Figure 32.

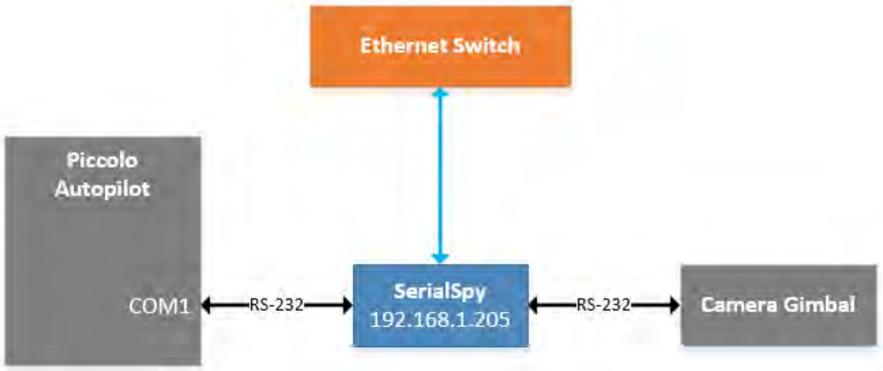


Figure 32. Simplified Block Diagram Showing SerialSpy Usage in a UAV Sentinel

The SerialSpy was specifically designed to overcome latency issues that arose with other methods of monitoring and modifying serial data. Previous design iterations of a serial communications monitor had used a Raspberry Pi single board computer and two USB to RS-232 adapters. A program written in C would monitor the serial ports and pass data between them passively in addition to modifying or blocking data as required. This approach had several disadvantages. First, the Raspberry Pi takes time to boot up before any such software could run, on the order of 30 seconds to 1 minute. This means that the autopilot and gimbal would be unable to communicate until the Raspberry Pi had fully booted and launched the software. Additionally, there is a certain latency and overhead associated with this approach. Each USB to serial converter adds several milliseconds of delay, and the Linux kernel also adds some delay. Although the processing time of the software is negligible, it must wait for the entire serial data packet to be received before re-transmitting the packet out the other RS-232 port.

The SerialSpy runs on a bare-metal microcontroller with no operating system, and so it boots up in milliseconds and immediately allows data to flow passively between the RS-232 ports. When not actively monitoring the data packets, the SerialSpy is able to pass data between the serial ports one byte at a time, effectively eliminating the delay. It does not have to wait and receive an entire serial packet before beginning to transmit data on the other serial port. For its RS-232 interfaces the SerialSpy uses DB9 connectors. P1 is a DB9-female connector wired as standard Data Terminal Equipment (DTE). P2 is a DB9-male connector wired as standard Data Circuit-terminating Equipment (DCE). The opposite gender connectors and appropriately crossed transmit/receive lines allow the SerialSpy to be inserted between two RS-232 devices. The SerialSpy can use baud rates from 1200 baud to 115200 baud.

Communications with the SerialSpy are provided by a USB-serial interface with a micro USB connector and an Ethernet interface with a standard RJ-45 connector. Both of these interfaces can be used to control and configure the SerialSpy. Commands include switching the device between passive and active mode, configuring the RS-232 baud rate, monitoring serial traffic in either direction, and configuring the various packet analysis modes for active mode.

As previously noted, the SerialSpy has two operational modes: passive and active. Passive mode will pass all data between the RS-232 ports without modifying or blocking any data. This mode is protocol agnostic; any format of data can pass through and be monitored. Active mode enables the SerialSpy's data processing module, which can modify or block serial data. Active mode is protocol-dependent and requires specific code to be written for protocol analysis. In either mode, the data can be monitored through the USB port or over Ethernet.

For the UAV implementation of the Sentinel, the data passing between the autopilot and the camera gimbal used the Piccolo gimbal communications protocol[1]. This protocol defines the specific packet structure of the data which includes a start-sequence header and length byte for each packet. The SerialSpy can be configured to be strict and only allow valid packets to flow through, or to allow all packets through, even if they are malformed.

For defense against cyber-attacks the SerialSpy was configured to search for specific packet types. Table 1 lists the packet types that were analyzed.

Table 4. Commands that the SerialSpy searched for in Gimbal Communications

Group	Cmd	Description	Actions
0x10	0x47	Gimbal telemetry	Compare GPS coordinates to truth data, alter if needed
0x10	0x10	Host GPS Data	Compare GPS coordinates to truth data, alter if needed
0x10	0x40	SPOI command	Block or alert when detected
0x00	0x45	SPOI command	Block or alert when detected
0x00	0x46	SPOI command	Block or alert when detected
0x00	0x43	Gimbal retract	Block or alert when detected

Messages that contained GPS coordinates from the autopilot were compared to known "truth data", which was provided over Ethernet from the SiCore computer portion of the onboard Sentinel. If the GPS coordinates were detected to be greater than 100 meters from the truth data, an alert was sent to the Sentinel. Optionally, the correct GPS coordinates provided by the SiCore computer could also be inserted into the datastream for the camera gimbal to use as metadata.

To defend against SPOI attacks and gimbal retract attacks, the operator could elect to have commands for SPOI and gimbal retract either completely blocked from being transmitted or to have an alert sent to the Sentinel over Ethernet

[1] Vaglienti, B. (2011). *Gimbal Communications v2.2.0.d.* Cloud Cap Technology

Appendix B: Gimbal Defenses

The following algorithms use structures and methods from the software development kit provided by Cloud Cap for ViewPoint plugin creation and are aimed at detecting the attacks found most feasible from section 4.2.2. Despite the following algorithm being written using an SDK, one could decode the information byte-wise from the message streams and follow the same algorithms.

Packet Detection

The method LookForGimbalPacketInQueue() searches through a queue of packets and determines if a packet of gimbal type (i.e., a gimbal packet) is present in the message stream. It then stores this packet in a predefined buffer. The packet is then inspected to see if the packet type is a gimbal command. All of the vulnerabilities in section 4.2.2 fall into this type with the exception of the user warning packet.

Retract/Deploy Command Detection

The gimbal packets are further inspected to determine if the packet group is that of *Gimbal command and control group*. If so, then it is passed to the method that checks if it can be decoded into a retract/deploy struct pointer. If the method returns false, the packet is ignored and the monitoring of the stream for packets continues. If the method returns true, then the stream is decoded into information determining whether the gimbal is being commanded to either retract or deploy.

Under the assumptions that normal operations would entail the retraction and deployment of the gimbal directly after take-off and directly before landing, the velocity of the gimbal relative to Earth and the distance of the gimbal from the ground station should be relevant criteria to determine whether the gimbal retract/deploy command appears to be authentic.

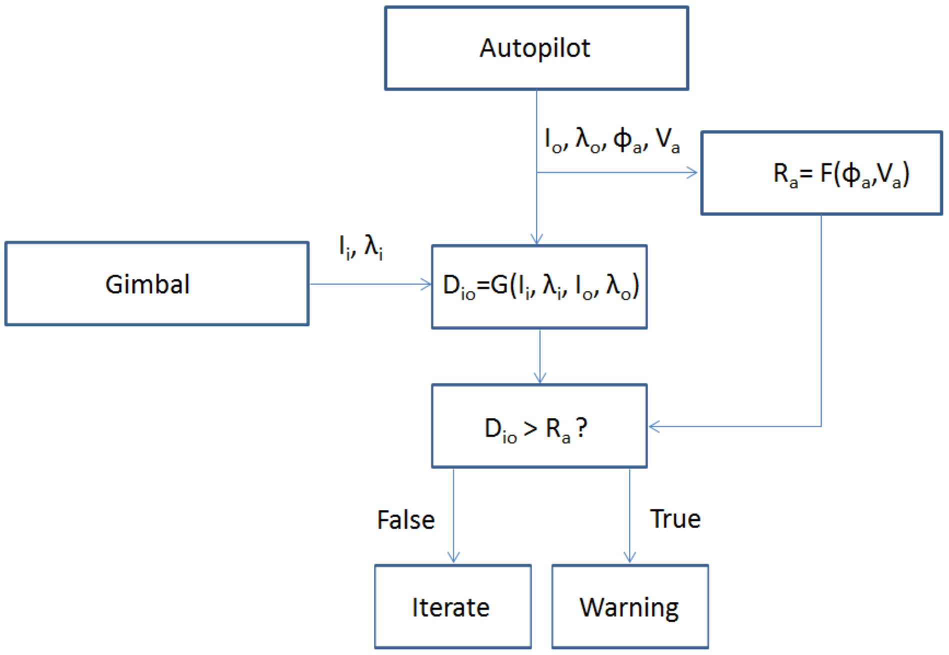
The aircraft velocity and position can be determined by monitoring the gimbal telemetry stream for packets of type HOST_GPS_DATA_GIMBAL_PKTTYPE and of group GIMBAL_POSITION_INFORMATION_GROUP. These telemetry packets can be decoded to give the GPS position and velocity of the aircraft. These two pieces of information can be used to determine what phase of flight the aircraft is in. If the phase is take-off or approach/landing, then the retract/deploy command is considered authentic. If the aircraft is in cruise or loiter mode, then the retract/deploy command should be considered malicious.

Erratic Gimbal Command Detection

To protect against a gimbal command attack it is assumed that during normal operations the gimbal should never be slewed to view a location above the horizon. Similar to the process just described, the telemetry stream is checked for gimbal packets in the queue. The method DecodeGimbalCmdPacket() is used to give an elevation angle of the gimbal. The aircraft's altitude can be determined from the GPS position data. If the aircraft altitude is higher than the altitude at which the gimbal is pointed, then the command is authentic. If the gimbal is pointed at a higher altitude than the aircraft then the command is considered malicious and the user can be warned via a message sent through a payload message stream using the autopilot command and control link.

Further constraints can be placed on the gimbal angles by limiting the gimbal orientation based on mission CONOPS. For example, if the UAV mission is to loiter overhead a specified target then the gimbal field of view should never extend outside the orbit of the aircraft.

A simple diagram (Figure 33) illustrates the application of mission context to authenticate the validity of the gimbal commands. With this paradigm, if the gimbal deviates from its intended use, then a warning message will be sent to warn the operator. The assumption is that when the aircraft is orbiting a location, the area being imaged by the camera should be within the radius of the aircraft's orbit. As shown in the figure, the image centroid location is determined by the function G which calculates the distance of the image centroid to the orbit center using the latitude and longitude of the gimbal and orbit centroid. This is done using an ellipsoidal Earth (WGS-84) model and calculating the distances using Vincenty's formulae.



l_i, λ_i : latitude, longitude of image
l_o, λ_o: latitude, longitude of orbit center
R_a : aircraft orbit radius
D_{io}: distance of image to orbit center
V_a : aircraft velocity
ϕ_a : aircraft bank angle

Figure 33. Gimbal Command Authentication

The radius of the aircraft's orbit (R_a) in steady, level flight can be found from its relationship to the aircraft's bank angle (⍰$_a$), velocity ($_{\mathrm{va}}$),

$$R_a = \frac{V_a^2}{g * \tan(\phi_a)}$$

If the centroid of the image lies outside the radius of the orbit, the operator will be warned that the gimbal pointing command does not appear to be authentic.

Appendix C: Schematic Diagrams

Figure 34 illustrates the circuit used to provide over discharge protection to the payload batteries. This circuit also allows the Piccolo to disconnect power to the payload. In the situation where the payload malfunctions or has some deleterious effect on other aircraft systems, it can be powered down. This also provides the ability to remotely restart the payload and return it to a known state.

Figure 35 is a schematic of the complete payload installation. The payload includes elements to simulate various attacks and is networked using a typical LAN topology. External to the payload enclosure is an Ethernet radio permitting communications with the payload ground station.

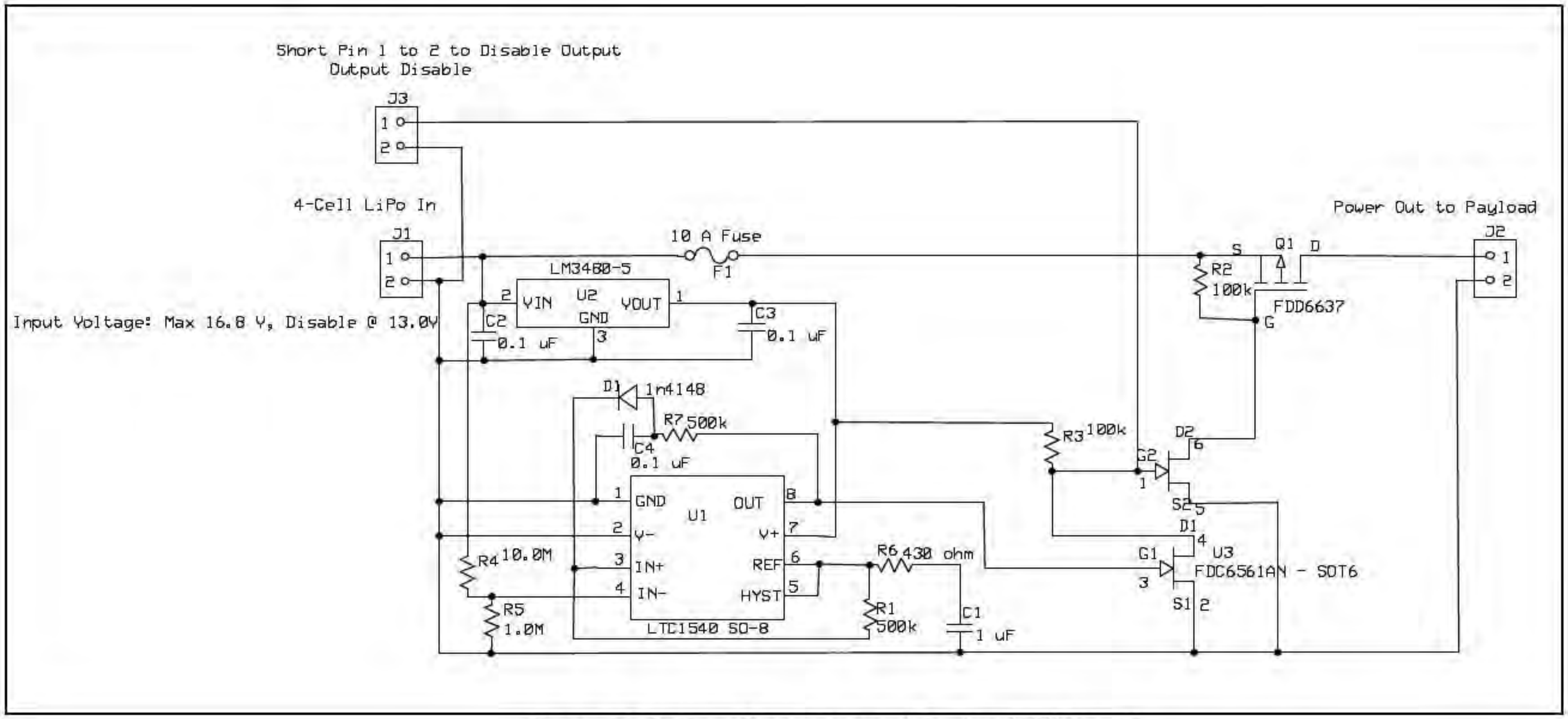


Figure 34. Battery Protection and Payload Kill Circuit

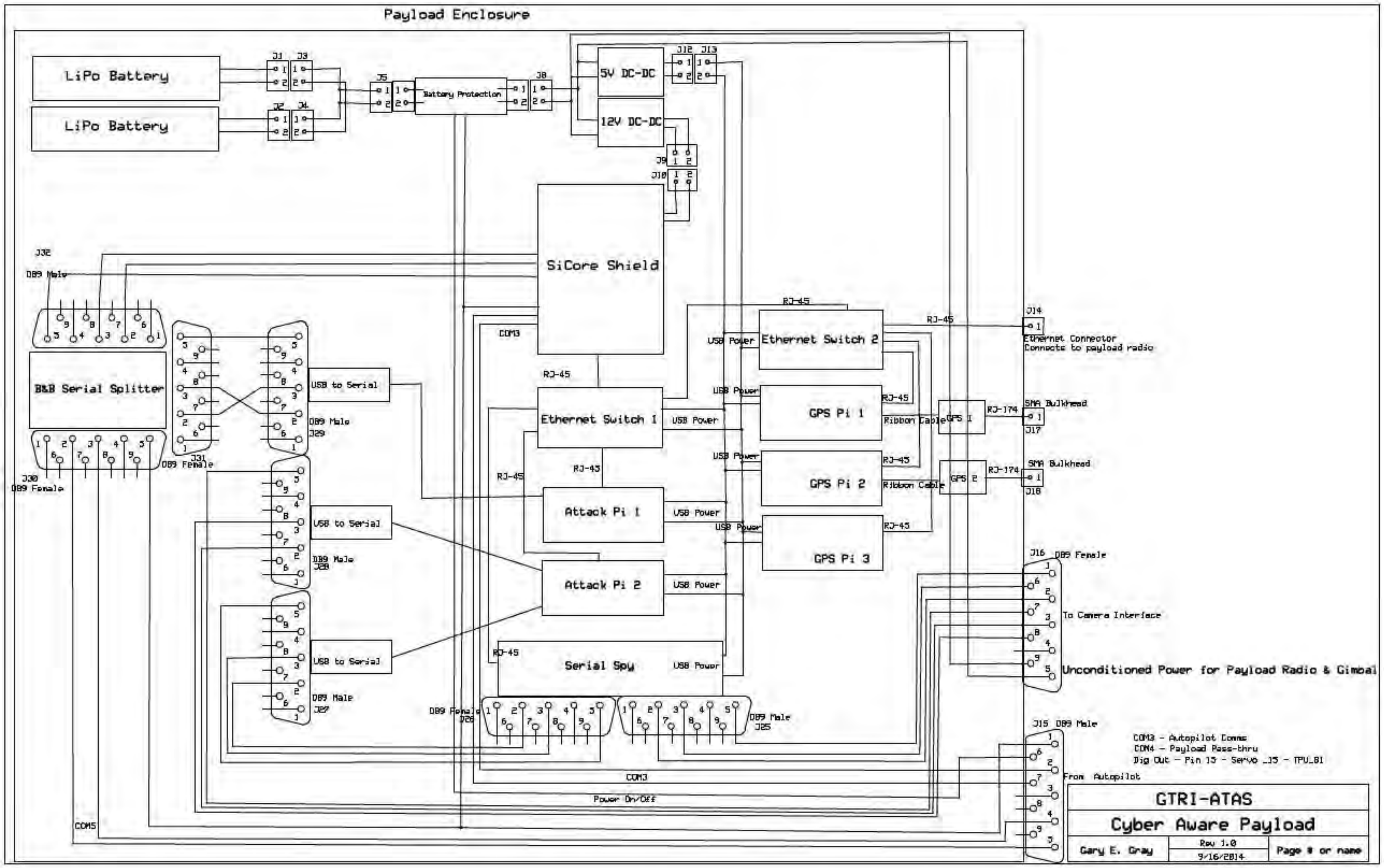


Figure 35. System Aware Payload Schematic

Appendix D: COA Information

Figure 36. COA Area of Operations

Figure 37. COA Area of Operations in Detail

Appendix E: Microhard Radio Vulnerability

Introduction

GAUSS' primary radio link is the MHX-910 (910 MHz) frequency hopping radio by Microhard Systems Inc. Microhard also offers 2.4 GHz radio systems which have been used on other GAUSS missions and are popular among other commercial-grade UAVs. GTRI researchers obtained several MHX-2420 2.4 GHz frequency-hopping radios (Figure 38) and were able to reverse-engineer the radio frequency (RF) protocol used to send data between the ground station and the UAV using software-defined radio (SDR) techniques and decompiling the radio's firmware. Although the Microhard MHX2420 radios use a different frequency, the modulation and hoping techniques and protocol are similar on the 910 MHz system.

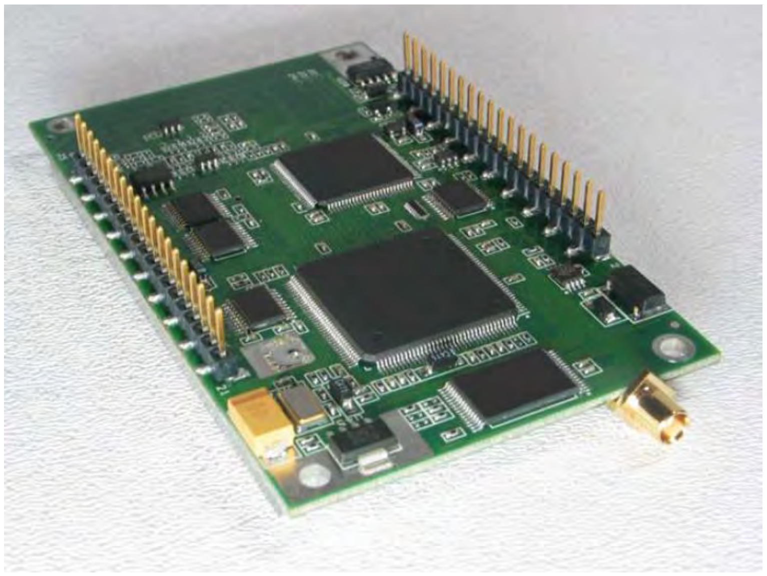


Figure 38. Microhard MHX2420 radio module

UAV Data Link

The RF data link is a pair of Microhard radios that connects a ground station to the UAV with a transparent serial connection. Each Microhard radio has a serial port. The ground radio's serial port is connected to the ground station, and the UAV's radio is connected to the autopilot. These serial ports are connected over an RF link established between the Microhard radios. The radios are configured independently from the UAV systems; the UAV software has no contextual information about the status of the link. This leaves open several vulnerabilities. Since the serial communication protocol between the ground station and the autopilot is known, simply being able to monitor the RF datalink would provide information about the aircraft's position, status, and payload data. Additionally, there is a possibility of injecting malicious commands or hijacking the datalink entirely so that a rogue ground station could assume complete control of the UAV.

The Microhard radios are configured over an additional serial port on the radio. For UAV use, the radios are typically configured as a master and slave radio.

Radio Analysis

GTRI researchers used a proprietary software-defined radio (SDR) system that is able to dynamically monitor and analyze large amounts of RF spectrum. The documentation for the Microhard radio indicates that the radios use frequency hopping spread spectrum (FHSS) within the 2.4000-2.4835 GHz spectrum.

For initial analysis, one MHX2420 radio was configured as a master and the slave radio was not powered up. Using the SDR, it was observed that the master radio broadcasts a "beacon" packet.

Additional information was gathered by downloading the radio firmware from Microhard's website and decompiling the firmware in order to learn more about the radio's functionality.

The beacon signal from master is broadcast at the beginning of each hop interval. The slaves transmit in timeslots within the overall interval thereafter.

The beacon signal uses BFSK modulation at a data rate of approximately 172776 bps. The main lobe is approximately 200 kHz wide and mark and space frequencies are roughly +-86388 kHz. Figure 39 and Figure 40 show spectral properties of a single beacon dwell that has been tuned down to a baseband center frequency of approximately 18 kHz. Figure 41 shows the FM demodulated output (green trace) of the dwell and the resulting symbols transmitted therein if a rate of 172776 bps is used.

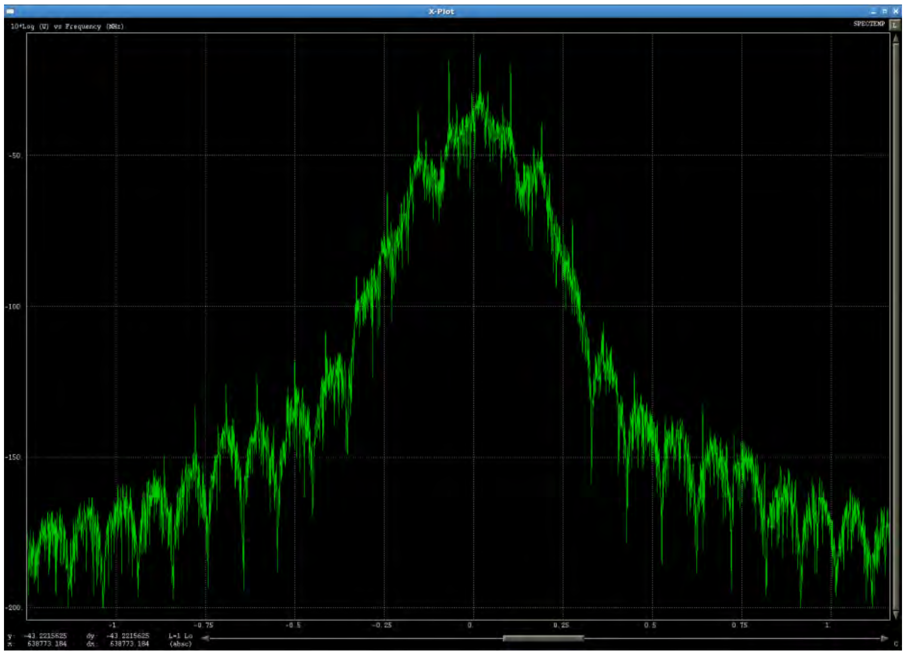


Figure 39. Spectrum Plot of Beacon Dwell Centered at 18 kHz

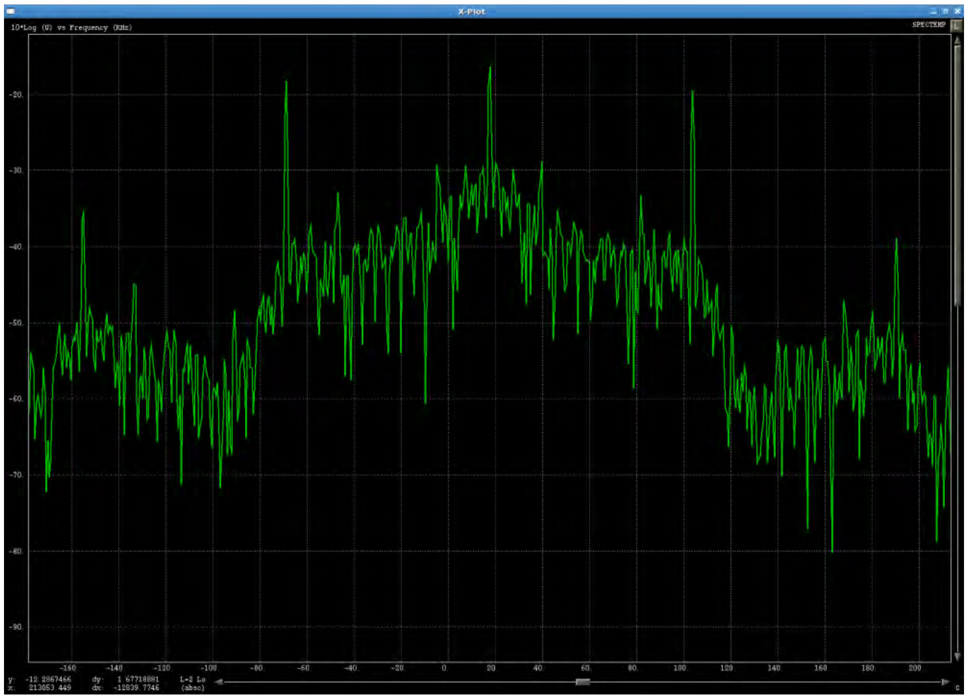


Figure 40. Zoomed In Plot of Beacon Dwell Centered at 18 kHz

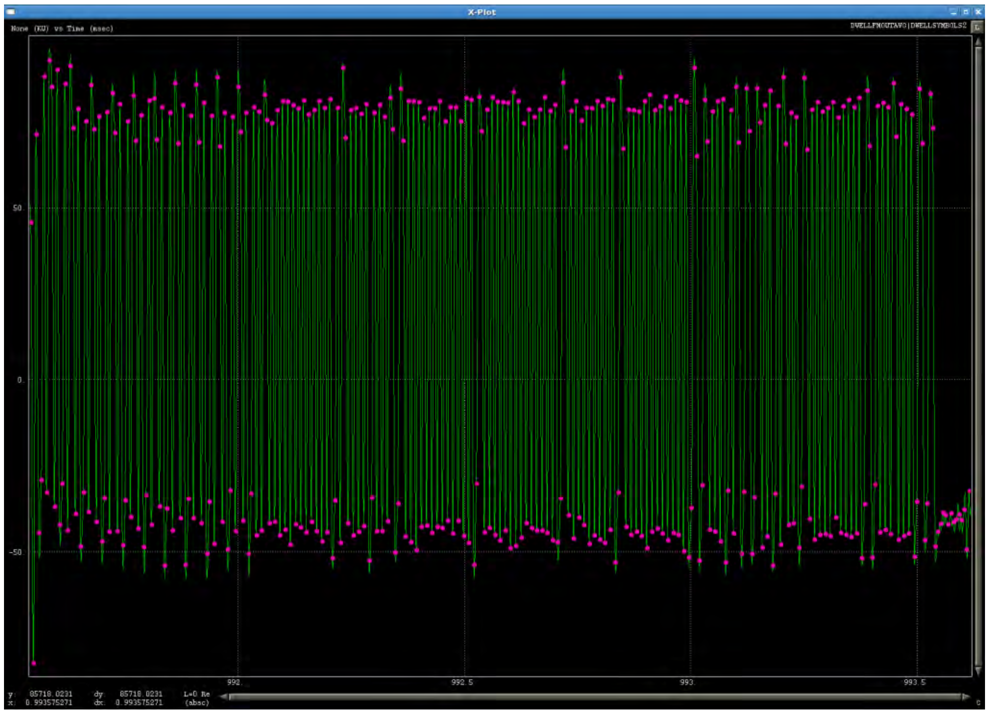


Figure 41. Overlaid Plot of FM Demodulated Audio (Green) and Assigned Symbols (Pink) of Beacon Dwell

Hop Structure

According to the MHX2420 manual, a total of 49 hopsets are available for use, the last 5 of which may be user-defined. Each hopset contains exactly 76 channels, and each repeats a 76-hop pattern repetitively. The base 76-hop pattern for all 49 hopsets was identified within the decompiled firmware. Each is listed in terms of the overall channel number instead of the actual frequency value. Channel numbers represent each consecutive 400 kHz channel, where channel 1 = 2.4016 GHZ and channel 202 = 2.4820 GHz. Although there are 202 total available channels, no hopset uses any channel higher than 191 (2.4776 GHz). However, the user may change this for hopsets 44-48 if desired.

Bit Decoding

1. The beacon transmits approximately 359 bits per dwell. Symbols must first be digitized into bits, and we assigned the high frequency a ‘1’ and the low frequency a ‘0’.
2. Over-the-air bits must be Manchester decoded. If assigning mark/space frequencies as just mentioned, then a 0b10 bit pair is a Manchester decoded ‘1’ and a 0b01 bit pair is a Manchester decoded ‘0’.
3. Manchester decoded bits must be grouped into 8-bit bytes, where the LSB of each byte was transmitted first (on the left side if incoming bits are plotted on a horizontal row). There are 8 possible byte alignment locations, so a receiver should scan through all of them.
4. The beacon transmits a fixed preamble of either 0x66666666E60060 or 0x60666666E60060. The byte alignment starting position from step 3 that produces a match with this preamble is the one that should be used for proper dwell alignment. Both preambles were observed in practice, and with the sole discrepancy being the second nibble in the sequence, the first two nibbles could be disregarded, with the remaining 6 bytes being used instead.

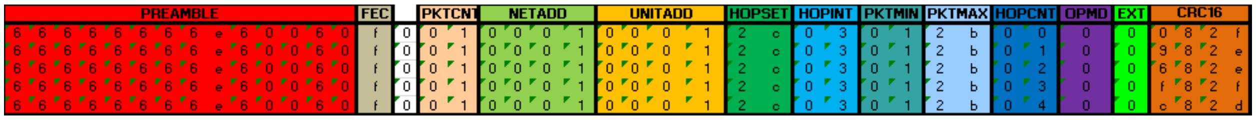
OTAR	1 0	0 1	1 0	1 0	0 1	0 1	1 0	1 0	0 1	0 1	1 0	1 0	0 1	0 1	1 0	1 0	0 1	0 1	1 0	1 0
Manchester Decoded	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
8 Possible Byte Start Locations	CD								CC								C			
		66								66										
			33								33									
				99								99								
					CC								CC							
						66								66						
							33								33					
								99								99				

Figure 42. Example decoding process for a single over-the-air dwell

RF Packet: Beacon Dwell Bit Structure

In order to properly decode the messages, it is important to determine the bit structure of each RF packet. Each beacon dwell transmits a total of 21 bytes, including the 7-byte preamble. After the preamble are the following bit fields:

1. FEC Identifier – 4-bit nibble
2. Unknown 4-bit nibble – likely simply a fixed '0' frame start identifier
3. Packet Count – 1 byte
4. Net Address – 2 bytes
5. Unit Address – 2 bytes
6. Hopset – 1 byte
7. Hop Interval – 1 byte
8. Packet Minimum Size – 1 byte
9. Packet Maximum Size – 1 byte
10. Hop Counter – 1 byte
11. Operating Mode – 4-bit nibble
12. Extension Flag – 4-bit nibble
13. CRC16 value – 2 bytes (Calculates CRC on data from Packet Count – Extension Flag)



PREAMBLE	FEC		PKTCNT	NETADD	UNITADD	HOPSET	HOPINT	PKTMIN	PKTMAX	HOPCNT	OPMD	EXT	CRC16
6 6 6 6 6 6 6 6 e 6 0 0 6 0	f	0	0 1	0 0 0 1	0 0 0 1	2 c	0 3	0 1	2 b	0 0	0	0	0 8 2 f
6 6 6 6 6 6 6 6 e 6 0 0 6 0	f	0	0 1	0 0 0 1	0 0 0 1	2 c	0 3	0 1	2 b	0 1	0	0	9 8 2 e
6 6 6 6 6 6 6 6 e 6 0 0 6 0	f	0	0 1	0 0 0 1	0 0 0 1	2 c	0 3	0 1	2 b	0 2	0	0	6 8 2 e
6 6 6 6 6 6 6 6 e 6 0 0 6 0	f	0	0 1	0 0 0 1	0 0 0 1	2 c	0 3	0 1	2 b	0 3	0	0	f 8 2 f
6 6 6 6 6 6 6 6 e 6 0 0 6 0	f	0	0 1	0 0 0 1	0 0 0 1	2 c	0 3	0 1	2 b	0 4	0	0	c 8 2 d

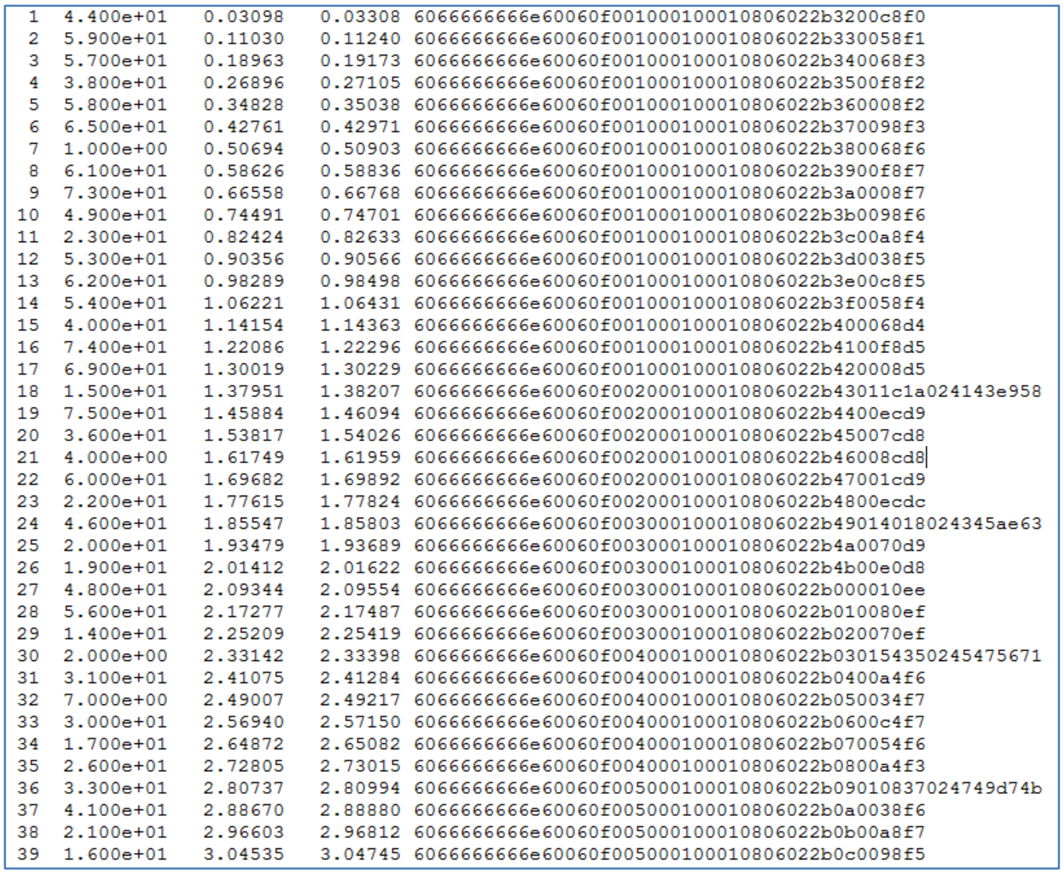
Figure 43. Interpreted bit field of beacon dwell RF packet example

RF Packet: Master Serial Data Bit Structure

When data is sent from a master, it is appended directly to the end of the aforementioned Beacon signal with no dead time in between. If the encryption key is set to 0, no encryption is used (see next section, "Encryption"). With an unencrypted data transmission, the Master dwell structure is the following:

1. Full Beacon as defined previously. The Extension flag will be a 0x1 when data follows that is sent directly from a Master as opposed to 0x0 when transmitting a Beacon only.
2. Directly after the CRC16 in the base Beacon, a 1-byte length field is transmitted
3. ASCII values of the transmitted data follow. The "Packet Minimum Size" parameter determines how many characters are transmitted per dwell. When simply holding down a key from an attached serial PC, the minimum number of characters are always sent in a packet. If transmitting a file, the "Packet Maximum" would likely be sent, although I did not test this.
4. A second 2-byte CRC16 is appended after the serial data. This CRC16 is calculated from all dwell data between the Packet Count and the final serial character using the same algorithm.

Data dwells are transmitted asynchronously within regular Beacon transmissions, and each packet may be repeated over a number of consecutive dwells according to the "Packet Retransmissions" parameter (register S113). Figure 44 shows an example data raster of a master transmission – dwells containing serial data are easily identified by their extended length.



```
 1  4.400e+01   0.03098   0.03308 6066666666e60060f00100010001080602 2b3200c8f0
```

Figure 44. Data Raster of Master Transmission of Data

Encryption

The encryption scheme employed by the system is nothing more than a simple XOR operation with the encryption key. Encryption is applied to the serial data characters only, and not to any portion of the base beacon signal or the length parameter of a data transmission. Encryption is performed by XORing each 2-byte pair with the 16-bit encryption key selected by the user. If only one 8-bit character is available, it is XOR'd with the MSbyte of the encryption key. Analysis of the binary representation for the power-of-two keys shows the clearest view of the simple XOR operation.

CRC Calculation

A 2-byte CRC16 value is appended both to the end of a regular Beacon portion of a dwell and after the extended serial data portion of the dwell as well. Both CRC's are calculated according to the standard CRC16 CCITT algorithm using a polynomial of 0x8005. The encryption key serves as the initial seed value for each CRC calculation, so varying the encryption key changes the checksum for a given set of data. For the beacon portion of the checksum, the CRC is calculated over the last 11 bytes of the dwell (consisting of the packet count through the extension flag). When serial data is appended to the end of a dwell, another checksum is also calculated over all data (including the first beacon CRC value) from the packet count through the last serial byte.

Forward Error Correction

All data presented previously dealt with the “Wireless Link Rate” set to a value of 2, which turns off Forward Error Correction (FEC). Setting this to a value of 4 incorporates the use of FEC in the transmission. Analysis of the beacon data with FEC enabled shows that the difference is a simple 4-bit to 8-bit recoding via a lookup table. All bytes from the packet count through the final CRC are simply remapped according to the lookup table listed below. FEC is applied as the last step in the transmission chain.

Unencoded	FEC Encoded
0	10
1	b1
2	52
3	b3
4	b4
5	55
6	b6
7	87
8	78
9	49
a	aa
b	9b
c	9c
d	ad
e	4e
f	7f

RF Packet: Slave Dwell Bit Structure

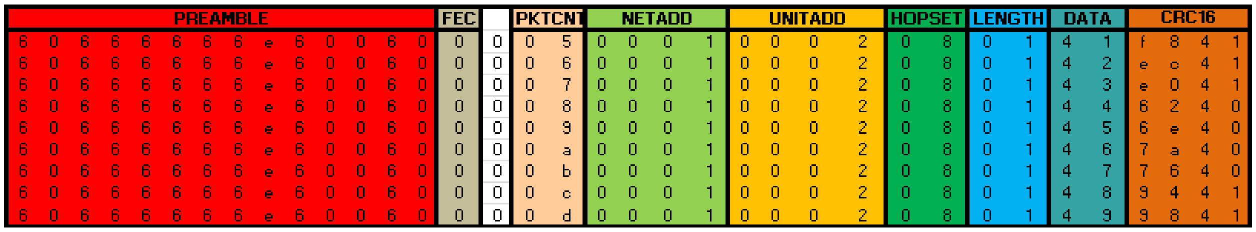
Minimal time and effort were spent on analyzing the slave data since the emphasis was on master signals. However, from brief data captures the dwell structure of the slave seems apparent:

Slaves use an identical 7-byte preamble as the master. Following the preamble are the following bit fields:

1. FEC Identifier (?) – 4-bit nibble. This field showed a value of ‘0’ for a slave, whereas it showed a ‘f’ or ‘a’ for FEC or non-FEC master transmissions
2. Unknown 4-bit nibble – likely simply a fixed ‘0’ frame start identifier
3. Packet Count – 1 byte
4. Net Address – 2 bytes
5. Unit Address – 2 bytes
6. Hopset – 1 byte
7. Data Length – 1 byte
8. Serial Data – Size ranges from packet min to packet max parameters
9. CRC16 value – 2 bytes (Calculates CRC on data from Packet Count – End of Data)

Since the slave only transmits when serial data arrives (i.e. it sends no fixed beacon), it always includes a length byte and serial data thereafter. Unlike the master dwell, there is only one CRC value, and it is calculated in an identical fashion as that of the master from the packet count to the final data character.

An example of the characters A-I being transmitted is shown in Figure 45. Since the packet min was set to 1 character, dwells send only one character at a time. Although encryption and FEC parameters were not varied for slave data, it is likely that they use the same algorithms as those previously discussed for master transmissions.



PREAMBLE														FEC		PKTCNT		NETADD				UNITADD				HOPSET		LENGTH		DATA		CRC16			
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	5	0	0	0	1	0	0	0	2	0	8	0	1	4	1	f	8	4	1
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	6	0	0	0	1	0	0	0	2	0	8	0	1	4	2	e	c	4	1
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	7	0	0	0	1	0	0	0	2	0	8	0	1	4	3	e	0	4	1
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	8	0	0	0	1	0	0	0	2	0	8	0	1	4	4	6	2	4	0
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	9	0	0	0	1	0	0	0	2	0	8	0	1	4	5	6	e	4	0
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	a	0	0	0	1	0	0	0	2	0	8	0	1	4	6	7	a	4	0
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	b	0	0	0	1	0	0	0	2	0	8	0	1	4	7	7	6	4	0
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	c	0	0	0	1	0	0	0	2	0	8	0	1	4	8	9	4	4	1
6	0	6	6	6	6	6	6	e	6	0	0	6	0	0	0	0	d	0	0	0	1	0	0	0	2	0	8	0	1	4	9	9	8	4	1

Figure 45. Example RF packets of slave dwell bit structure with characters A-I being transmitted

Conclusion

At this point we know sufficient enough information about the master Beacon signal that we should be capable of both deciphering it and replicating it in a custom system. By monitoring a single frequency channel, a sample Beacon dwell could be obtained. The structure of this dwell would tell us if FEC is applied or not, and it could then be easily converted to its non-FEC equivalent. At this step the beacon data from packet count to extension bit could be used along with the successive CRC value for obtaining the encryption key used. By running a brute-force CRC analysis over all 65536 possible initial seed values, the seed value that produced the observed CRC with the observed data is the encryption key in use. Once this is determined any additional serial data that may be transmitted from a master may also be deciphered. Finally, since all hopsets follow a repeating pattern, the hop count value in the dwell could be cross-checked along with the actual frequency channel in use to potentially identify the exact hopset in use as well. If this single dwell was insufficient to identify the lone hopset (i.e., multiple hopsets use the same frequency during the same hop dwell), reception of only a few more dwells should be sufficient to isolate the set. Since the preamble is a fixed data pattern and the remaining Beacon bits are known, full knowledge of the hopset and encryption key would allow for a custom system to fully generate any valid Beacon signal of interest.

By implementing the above attack methodology, a system that monitors or injects data packets into a live UAV datalink is definitely a plausible attack vector.

SYSTEMS ENGINEERING
Research Center

Security Engineering Project

Part 2 – Human Factors Engineering and System-Aware Cybersecurity

Copy available by request to the government

University of Virginia

Chris Gay
Barry Horowitz
Nathan Lau (formerly; currently Virginia Tech)
Kevin Leach

The MITRE Corporation

Michael Dinsmore
Chris Jella
Michael Lewis
Darren Neal
Jarret Rush

Technical Report SERC-2015-TR-036-4 Part 1
January 31, 2015

SYSTEMS ENGINEERING
Research Center

Security Engineering Project

Part 3 – Cloud Architectural Assurance and Protecting Systems with Cloud-based System-Aware Methods

Principal Investigator: Dr. Barry Horowitz, University of Virginia

Senior Researcher:

Dr. Marty Humphrey, Associate Professor, University of Virginia

Technical Report SERC-2015-TR-036-4 Part 3

January 31, 2015

Executive Summary

The Systems Engineering Research Center (SERC) has developed a novel cybersecurity concept for embedding security solutions into systems called *System-Aware Cybersecurity*. The overall goal of the System-Aware program is to develop low cost methods of protection against cyber exploits by our adversaries. Working through the SERC, the University of Virginia (UVa) and the Georgia Tech Research Institute (GTRI), Phase 1 efforts of the program were focused on advancing the System-Aware Cybersecurity concepts and evaluating a number of specific security design patterns that were intended to be reusable across a variety of applications. The major goal Phase 2 goal of the Sentinel program was to demonstrate the feasibility of System-Aware Cybersecurity design patterns designed in Phase 1 and to demonstrate the capabilities of a physical version of those protections to protect a system in a live environment and to experiment with the protections in order to monitor selected critical system functions of that system – in this case, an unmanned aerial vehicle (UAV). We call the physical implementation of this novel protection the Sentinel. Furthermore, those critical system functions were identified and analyzed for system vulnerabilities using an architectural selection methodology developed in Phase I of the project.

Another goal of the Phase 2 project effort was a new Phase 1 effort to begin to evaluate two key areas surrounding cyber security protections and the use of private cloud infrastructures. The first area of our research focus was evaluating the use of the agile features of a cloud architecture as a platform for the delivery of System-Aware Cybersecurity protections to another system. Two use-case applications were the subject of evaluations in this project phase: 1) Experimentation of a prototype implementation of a simulated video surveillance system application in a private cloud architecture and the application and testing of System-Aware Cybersecurity security design patterns using the cloud infrastructure as a method to protect the application for an attack, 2) the Sentinel protection applications which were developed for the protection or the UAV system in the other part or our research efforts. A third effort is under way to implement a video imagery exploitation application called AIMES used by the Air Force and the Army to analyze video from manned and unmanned surveillance platforms (this portion is complete), and to apply System-Aware techniques to protect functions within that system using System-Aware methods running in the cloud. The attack scenarios and monitoring and detection of possible attacks is currently in design and will continue to be our focus for next year's activities.

In addressing the goal of evaluating the feasibility of securing the cloud infrastructure itself, we developed two new methods of assurance for the platform while utilizing the Openstack private cloud architecture as an example platform. First, we introduced the concept of cloud infrastructure reporting. The basic idea was these report notifications would provide experimental timing information as the baseline for normal (correct) private clouds operations; these baseline measurements would then be consulted as the basis for asserting anomalous behavior. The second concept we introduced was the use of active probes to evaluate the behavior of the cloud infrastructure. Specifically, we experimented with the delivery mechanism of resources for virtual machine creation. We designed a self-contained, new activity that was scheduled on-demand and whose functionality was to actively probe the infrastructure for anomalous behavior.

The concepts which we designed during this phase proved out that both the use of the cloud to provide an agile platform for Sentinel protections is feasible and adds value, and that new methods are feasible to begin to provide assurance to the cloud infrastructure through the development of new security design patterns for the cloud platform itself.

Table of Contents

1. Background 202
2. Research Activities.. 203
2.1. Investigating the Use of Private Cloud Platforms for Hosting Sentinel Application Components and Design Patterns.. 203
2.2. Implementation Feasibility of a Cloud Integrity Assurance Theory 204
2.2.1. Conceivable Persistent Threats.. 204
2.2.2. Research Questions addressed: 204
2.3. Potential Cloud Platforms 205
2.4. Cloud Simulation Environment 207
2.5. Cloud Test-bed Environment.. 207
2.5.1. Use-Cases: System-Aware Protections In The Cloud.. 208
2.5.1.1. Use-Case: Cloud-based System-Aware Protections of a Security Video Surveillance Platform 208
2.5.1.2. Use-Case: The UAV Sentinel Applications in the Cloud 210
2.5.2. Issues in Test-bed Configuration.. 211
2.5.3. Creating the AIMES Test-bed Environment.. 211
3. AIMES System– An Example of Applications in the Cloud and As a System to Apply System-Aware Techniques 212
3.1. AIMES Overview.. 212
3.2. The AIMES Test-bed Implementation 213
3.3. Virtualization for the AIMES Application Test-bed 213
3.4. Progression of Test-bed Configurations 214
3.5. Potential Attack Opportunities 215
3.6. Ensuring the Private Cloud Infrastructure.. 216
4. Future Work 219

List of Figures

Figure 1. An example of a diverse environment for delivering security to a system and its applications using design patterns implemented in multiple clouds: OpenStack (left) and Microsoft Hyper-V Cloud (right)........ 203
Figure 3 - Openstack Overview Diagram........ 207
Figure 3 - Cloud Implementation of System-Aware / Sentinel Protections on the UAV system functions 211
Figure 4 - Overview of AIMES System Architecture........ 213
Figure 5 - Diagram of possible attack vectors within the Openstack Infrastructure........ 217

1 Background

Cloud architectures, both private and public, are quickly becoming the platform of choice for system deployment for major systems due to their flexibility in assigning computing resources and the impact that flexibility has on costs of reallocating resources on an as-needed basis. The University of Virginia (UVA) has undertaken research activities investigating how to use the agility of cloud architectures to aid in cyber security by providing a platform for deploying System-Aware Cybersecurity design patterns for enhancing system protection. This effort was based largely on previous SERC research efforts conducted at UVA, mainly focused on developing and utilizing security design patterns to protect physical systems.

In our early efforts on cloud architectures, two specific areas of research interest have developed as new research questions, which we have expanded in the past year:

- Can we exploit the agility of cloud service architectures to serve and to enhance the platform for delivering Sentinel-based, cyber security protections?
- Can we build a control system-based theoretical framework for addressing how to protect the private Cloud Sentinel?

System-Aware Cybersecurity

The Systems Engineering Research Center (SERC) has been engaged with the Department of Defense (DoD) in developing a novel cyber security concept for embedding security solutions into systems; this new concept is referred to as *System-Aware Cyber Security*. These solutions provide greater assurance to the most critical system functions by providing an additional layer of defense that complements perimeter and network security solutions that serve to guard the entire system from penetration. System-Aware solutions are particularly effective at guarding against insider and supply chain attacks that circumvent perimeter security solutions. The broad objective of the System-Aware program can be thought of as reversing cyber security asymmetry from favoring our adversaries, to favoring the US; i.e., from favoring a small investment in straightforward cyber exploits to favoring small investments in System-Aware cyber security solutions for protecting critical system functions.

To-date, the SERC and a University of Virginia (UVa) led team, consisting of the UVa and the Georgia Tech Research Institute (GTRI), have advanced the System-Aware cyber security concept and evaluated a number of specific design patterns that are intended to be reusable across a variety of applications. These patterns include, but are not limited to, employing diverse redundant components in critical subsystems, using voting techniques across diverse redundant components for real-time discovery and elimination of infected components, dynamically modifying the configuration of software components in systems through virtual configuration hopping techniques, dynamically modifying the configuration of the hardware/software components in systems through physical configuration hopping techniques, using system specific data consistency-checking to determine if critical system information has been manipulated, and where applicable, use of analog components as trusted elements to perform critical security functions in systems. Furthermore, a decision support framework has been developed for use by systems engineering teams in selecting a subset of available design patterns for integration into a cyber security system architecture.

This portion of the project focuses attention on the use of private cloud architectures as both a host for System-Aware protections and methods for the assurance of private could architectures as trusted platforms.

2 Research Activities

2.1. Investigating the Use of Private Cloud Platforms for Hosting Sentinel Application Components and Design Patterns

As the Sentinel concept has been developed, certain security design patterns have emerged as candidates for implementation on a cloud platform. Specifically, diverse redundancy, secure voting and configuration hopping are candidate security design patterns for delivering Sentinel application services across private cloud architectures. During the past year's efforts, the cloud architecture's inherent capabilities to use multiple hypervisors working on different operating systems within a single private cloud infrastructure (e.g., OpenStack) or the ability to configure multiple cloud infrastructures (e.g., Microsoft Hyper-V Cloud Services in addition to an environment such as Openstack) as a way to provide a diverse environment for deployment of Sentinel functionality. The figure below illustrates an example of a diverse environment for adding security to a system through the use of monitoring and control design patterns implemented in multiple clouds serving as a Sentinel.

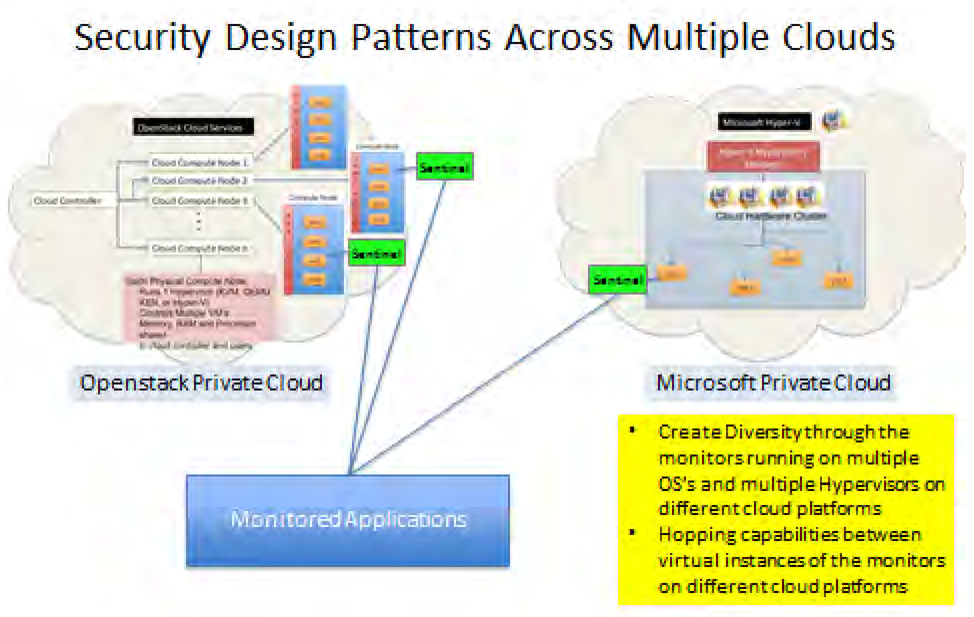

Security Design Patterns Across Multiple Clouds
OpenStack Cloud Services
Cloud Controller
Cloud Compute Node 1
Cloud Compute Node 2
Cloud Compute Node 3
Cloud Compute Node n
Sentinel
Sentinel
Sentinel
Openstack Private Cloud
Microsoft Hyper-V
Cloud Hardware Cluster
Microsoft Private Cloud
Monitored Applications
Create Diversity through the monitors running on multiple OS's and multiple Hypervisors on different cloud platforms
Hopping capabilities between virtual instances of the monitors on different cloud platforms

Figure 1. An example of a diverse environment for delivering security to a system and its applications using design patterns implemented in multiple clouds: OpenStack (left) and Microsoft Hyper-V Cloud (right).

There are many unanswered questions that need to be addressed as we began to look at cloud architectures as a part of a Sentinel-based, System-Aware protection mechanism. In our efforts this year in this area - leveraging the cloud environment in order to provide a more diverse and agile platform for the Sentinel protections, we have experimented with two different implementations of Sentinel protections. The first step required the implementation of a private cloud test-bed environment to host virtual computing platforms that would execute Sentinel monitoring and protection code. We developed an implementation of the Openstack private cloud architecture as a host for both the Sentinel protection algorithms and a place to run applications in the cloud for future experimentation. This environment is described in Section 2.5.

For our initial experiments with using the agility and diversity of the cloud architecture, we first implemented a sample video surveillance system in order to develop a proof-of-concept environment to test initial theories on implementing System-Aware Sentinel protections utilizing the agility of a cloud for for hosting cybersecurity projections (described in Section 2.5.1.1). We then chose to re-implement the same protections that were used in our unmanned aerial vehicle (UAV) project to protect critical flight control and payload system functions on board the aircraft (described in Section 2.5.1.2). As a more substantive use case, we have successfully implemented a working demonstration system of the (Advanced Imagery Exploitation System - AIMES) in order to experiment and test methods for securely protecting ground-based systems that are directly interacting with an airborne system (described in Section 2.5.3).

2.2. Implementation Feasibility of a Cloud Integrity Assurance Theory

The cloud architecture itself can be viewed as a control system as it is responsible for managing the delivery of computing resources to the applications and users that are requesting environments to support their functional requirements. This control system is vulnerable to cyber attacks. If the infrastructure of the cloud is compromised, then the applications supported by that infrastructure are also vulnerable. In this effort we investigated the feasibility of creating a theoretical security monitoring framework for assuring cloud infrastructure integrity and providing example implementations as a step in proving feasibility.

Our goals for this portion of the effort included 1) Developing an initial theoretical framework for assuring cloud infrastructure integrity, 2) Identifying both the opportunities to apply the theory and practical limitations regarding application of the theoretical framework, 3) Developing illustrative design patterns for implementing system integrity solutions, 4) Demonstrating selected design patterns using an off-the-shelf cloud infrastructure, including both attack detection and responses to those attacks.

2.2.1. Conceivable Persistent Threats

In considering a new security monitoring framework for cloud architectures, we must first investigate the potential risks inherent in the infrastructure itself and potential threats to the data that moves in and out of the systems hosted on the cloud platform:

- **Disrupting Cloud Management** - Attacking the management framework of the cloud infrastructure to introduce application latencies that impact military operations.
- **Data Barriers Within the Cloud -** Attacking the Cloud infrastructure so that it disrupts normal access to timely, mission critical data.
- **Data Barriers at the Cloud Edge** - Attacking the Cloud infrastructure so that it delays or denies data flow into or out of the Cloud storage environment for selected periods of time related to specific military operations.
- **Data Confidentiality** - Attacking the Cloud infrastructure to gain access to sensitive data that is co-located on the cloud system (adversary learns what you know).
- **Data corruption -** Attacking the Cloud infrastructure to create mechanism for insertion of arbitrary bit-errors for all data, so as to complicate use and response for selected or random periods of time

2.2.2. Research Questions addressed:

Private cloud platforms offer flexible architectures capable of delivering scalable, flexible and cheaper computing platforms in support of mission critical applications in our organizations. It is crucial that we understand the new cyber-security risks that are introduced by the cloud infrastructure and cloud management architectures that serve as hosts to those applications. Our research effort aims to investigate the security issues around the cloud and start to answer questions in four major areas:

The first questions focus attention on the use of the flexibility of a cloud to protect other cyber physical or information systems.

1. Can we leverage the diversity and flexibility of the cloud architecture and our previous research efforts into methods of utilizing security design patterns to further enhance the protection of command and control systems in mission critical applications?
2. Can we build a prototype and demonstration environment, through a specific use case such as the AIMES imagery exploitation system provided by Leidos, that can monitor and detect changes in the cloud architecture and provide a more "trusted" delivery platform for private cloud architectures and the applications that they ultimately host?

Additionally, questions are focused on the protections of the private cloud architecture itself.

3. What theoretical approaches can be developed in or adapted from other areas of computer science to monitor and to detect potential anomalous behavior in the cloud stack itself and ultimately protect its vulnerabilities? We aim to address issues including: What data needs to be collected and where is it sourced? What are the analytical techniques needed to leverage the data collected for monitoring purposes, detection of anomalous behaviors, and classifying detections? What are the potential responses to different classes of detections?
4. Can we build an approach for monitoring and for detection which will look at the at the cloud services stack from a "bottom-up" view? We will concentrate on how the cloud environment itself operates including its interactions with the hosting hardware environment first, and move up the cloud's operational stack addressing the specific applications that are running in the cloud and associated virtual machine environment as a last step. We will look very closely at how the cloud configuration itself interplays with the applications and how those interfaces can be monitored.

2.3. Potential Cloud Platforms

To begin the process of looking at the vulnerabilities of a cloud's architecture, we must first identify and fully understand the specific architecture that we wish to protect. Both commercial and governmental organizations are turning to the open source cloud software world to support the development of private cloud architectures for their enterprises. The primary drivers for this are the low cost entry-point and the prospect for portability of applications across the enterprise. Three leading open source solutions come to the front for consideration: Openstack, Apache CloudStack and Eucalyptus.

The Openstack open source platform is maintained and developed through the OpenStack Foundation. The foundation is supported and sponsored through a vast network of individuals and companies (https://www.openstack.org/foundation/companies/) including IBM, HP, AT&T, Rackspace and Canonical – the main supporter for the Ubuntu Linux operating system. In fact, the OpenStack environment is included in the basic distribution of the Ubuntu OS. It is also a fully supported option for SUSE and Redhat versions of LINUX. Rackspace both provides extensive development efforts for the Openstack platform and also uses the environment for delivery of public cloud services. Rackspace and NASA were the original developers of the OpenStack environment. OpenStack has a broad and quickly expanding user base and a very deep pool of open source developers and seems to have momentum as the open source cloud platform of choice at this point.

The Apache CloudStack platform is a technology platform that was first developed in 2009, purchased by Cirtix in 2011 and released it through to the Apache Software Foundation (ASF) open source community in 2012 . The ASF is also supported by many individuals and companies (http://www.apache.org/foundation/thanks.html) that provide varying levels of development and monetary support. CloudStack is a more “polished” cloud delivery platform, especially in terms of installation. It is widely used and hosts many production application environments.

The Eucalyptus cloud architecture is the oldest of the open source cloud environments. It is a major cloud architecture partner with the Amazon public cloud service (Amazon Web Services). They too are supported by many corporate sponsors (https://www.eucalyptus.com/) – including Microsoft, HP, Google and Citrix. Because of the relationship that they have with AWS, they market their capabilities to offer a hybrid solution of private and public based clouds that enable easy migration to and from the public and private environments as needed by the user.

In the end, our research activities should apply generically to any of these platforms as they all ultimately deliver similar functionality in various cloud services while they each offer their own specific features, benefits and drawbacks for different implementations. However, for our purposes, we will need to focus on one platform for use as a testing, analysis and management platform for clouds. As we see that the OpenStack platform is getting a very large amount of attention from organizations wishing to move applications into open source, private cloud-based platforms, we will focus on the Openstack implementation for this project. If we find use cases that require other platforms, will include those in our analysis, as needed, including integrating other cloud services into the mix of diverse platforms in order to provide security to the protections and to the systems we are protecting using System-Aware techniques.

It should also be noted that there are other commercially focused private cloud solutions from companies such as Microsoft (Hyper-V based services) and the VMware solutions for virtualizing computing may play a role in this research, as well. Initially, the focus will be on open source solutions as they offer the most cost effective entry-point solution for organizations as they move into privately hosted clouds. In addition, these solutions are more commonly referred to as virtual computing environments and may not meet the definitions of private Infrastructure as a Service (IaaS) offering.

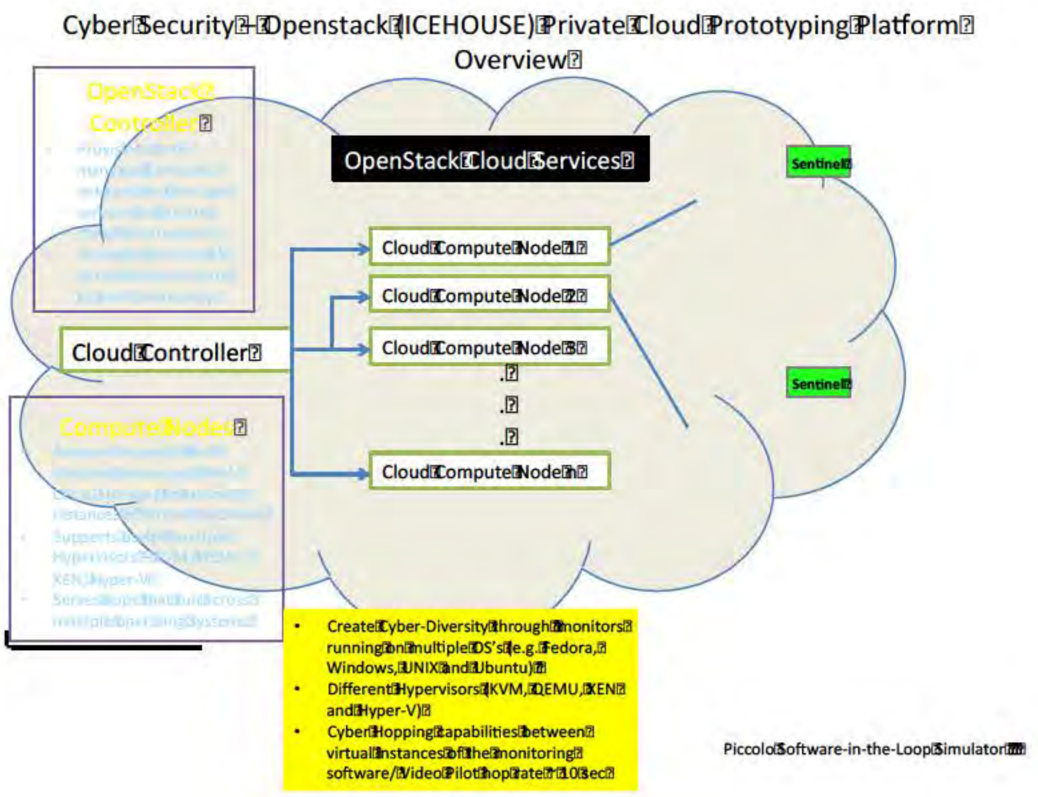

Cyber Security - Openstack (ICEHOUSE) Private Cloud Prototyping Platform Overview
OpenStack Controller
OpenStack Cloud Services
Sentinel
Cloud Compute Node 1
Cloud Compute Node 2
Cloud Controller
Cloud Compute Node 3
Sentinel
Compute Nodes
Cloud Compute Node n
Supports multiple Hypervisors (KVM, QEMU, XEN, Hyper-V)
Create Cyber-Diversity through monitors running on multiple OS's (e.g. Fedora, Windows, UNIX and Ubuntu)
Different Hypervisors (KVM, QEMU, XEN and Hyper-V)
Cyber Hopping capabilities between virtual instances of the monitoring software/Video Pilot hop rate ~10 sec
Piccolo Software-in-the-Loop Simulator

Figure 2 - Openstack Overview Diagram

2.4. Cloud Simulation Environment

We anticipate another important aspect of our research endeavors will focus on a need to create a cloud simulation and test-bed environment. This environment will be utilized for testing of new monitoring and detection techniques and also will be used to create test data sets that could serve as part of the monitoring and detection algorithms applied to physical cloud systems. We will attempt to find and to use functionality provided by existing simulators or emulators that will allow us to simulate and analyze components of the cloud architecture including the cloud configuration parameters, networking design within the cloud, hypervisors, virtualization control and management of storage, memory and processors. The Openstack environment is described further in Section 2.5.

2.5. Cloud Test-bed Environment

We implemented our private OpenStack (ICEHOUSE version) cloud using a set of scripts provided by StackGeek.com. These scripts automated the process of installing necessary packages and libraries, and making changes to various configuration files. In addition, a URL provided during setup facilitated the addition of new nodes to the cluster.

The current cluster consists of one controller node and one compute node. We have the option of adding more compute nodes as necessary. We initially decided during the setup process to exclude a node dedicated to Openstack Swift object storage, as it was not necessary for the direction of our cloud research. Compute nodes met our goal of providing an environment in which to potentially offload Sentinel protections currently isolated in Raspberry Pi single-board computers. However, as the research continues to progress, the Swift module will play a more important role. Particularly, Swift, which is the persistent, shared storage management system for the Openstack private cloud will needed to aid in the evaluation of security of the cloud infrastructure and the methods that are used to move and store data in and between virtual instances inside the cloud environment. It is anticipated that the Swift storage will become the main imagery and metadata repository for the AIMES system for experimentation during the follow-on phase of the project.

2.5.1. Use-Cases: System-Aware Protections In The Cloud

For our initial experiments with using the agility and diversity of the cloud architecture, we chose two different applications to test out the feasibility of running System-Aware Cybersecurity security design patterns in a cloud architecture and to see if we could leverage the flexibility and agility of the could to enhance those protections. The first application was a simulated video security surveillance system. The second was the Sentinel applications themselves, basically a re-implementation of the same protections that were used in our unmanned aerial vehicle (UAV) project to protect critical flight control and payload system functions on board the aircraft.

2.5.1.1. Use-Case: Cloud-based System-Aware Protections of a Security Video Surveillance Platform

This task was designed as an early, proof-of-concept step to demonstrate the capability to implement System-Aware Cybersecurity design patterns utilizing the agility and flexibility of a private cloud, our multiple private cloud architectures as a hosting platform for a few security design patterns. For this effort, we implemented two sample cloud environments, an implementation of an Openstack private cloud and an implementation of a Microsoft Cloud Services Cloud which is an environment configured to utilize the Hyper-V product-line to deliver on-demand virtual environments to used in a Microsoft Active Directory Domain.

The overall goal for project task: Investigate the use of private cloud platforms for hosting Sentinel application components and security design patterns. This was accomplished by:

- Enabling *Diverse Redundancy/Voting* and *Configuration Hopping* System-Aware Cybersecurity design patterns by delivering Sentinel application services for a sample system across a private cloud architecture utilizing:
  - Multiple hypervisors supporting the deployment of different operating systems within a single private cloud infrastructure (OpenStack).
  - Showing the addition of a second, diverse cloud service delivery platform, in this case Microsoft Hyper-V Cloud Services so the design patterns could be executed across multiple, diverse cloud infrastructures.

To test the ability to apply our test design patterns, we first needed an example application to test the effects of executing time-sensitive functions such as *System Configuration Hopping* and *Diversity with Secured Voting techniques.* The first test case chosen was a video surveillance server (shown in Figure 3) installed into the cloud system architectures. Every effort was made to make the environment as possible with the server components installed in different operating systems, spread to different hypervisors (such as QEMU, KVM, XEN and Hyper-V) and across cloud architectures (Openstack and Microsoft). We chose the video surveillance application specifically because latencies that were introduced into the system by the security design patterns would be very visible to the end users and would affect the smoothness of the video displayed to the end user.

We then implemented our design patterns. The *Configuration Hopping* design pattern required us to install multiple versions of our sample system and to implement a mechanism to show the System-Aware protections which were to *Hop* between those application environments and vote out bad streams either automatically or on command. We then developed a basic attack scenario, a piece of malware which enabled a video-replay attack available on command to infect the video server and make the live video stream repeat a segment of video, thus hiding activities from the live stream from the viewer. We developed a simple interface to show the attack, and then enable the protections System-Aware protections and show the attack again with the protections in place. Careful efforts we designed into the demonstration to show the ability to hop between system functions without having latencies impact the surveillance system viewer's experience. Much of this work was based on Rick Jones' Masters thesis – 2011. (Figure 4 shows an overview of the system and the System-Aware protections in place).

In this demonstration system, we were able to successfully demonstrate the ability to execute the selected design patterns in a very diverse, cloud-hosted architecture. This work has laid the cornerstone for the other efforts undertaken as part of the cloud-based project.

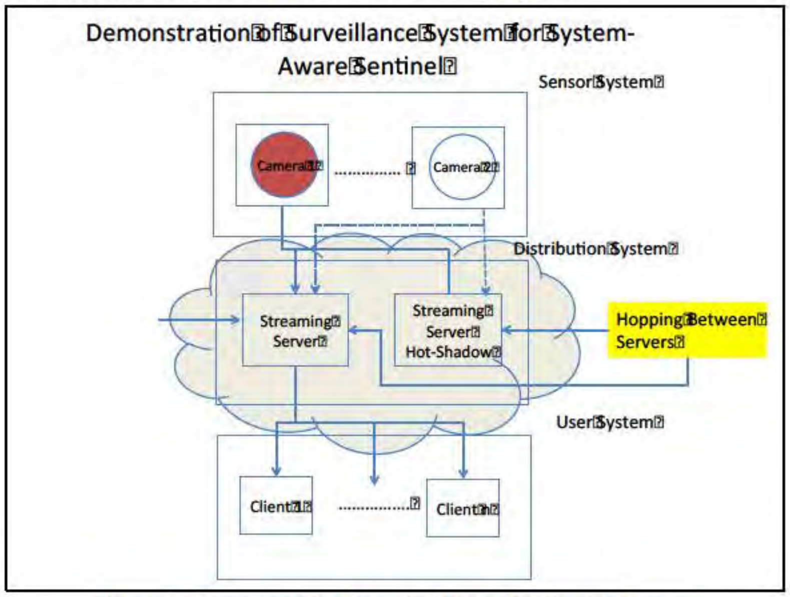
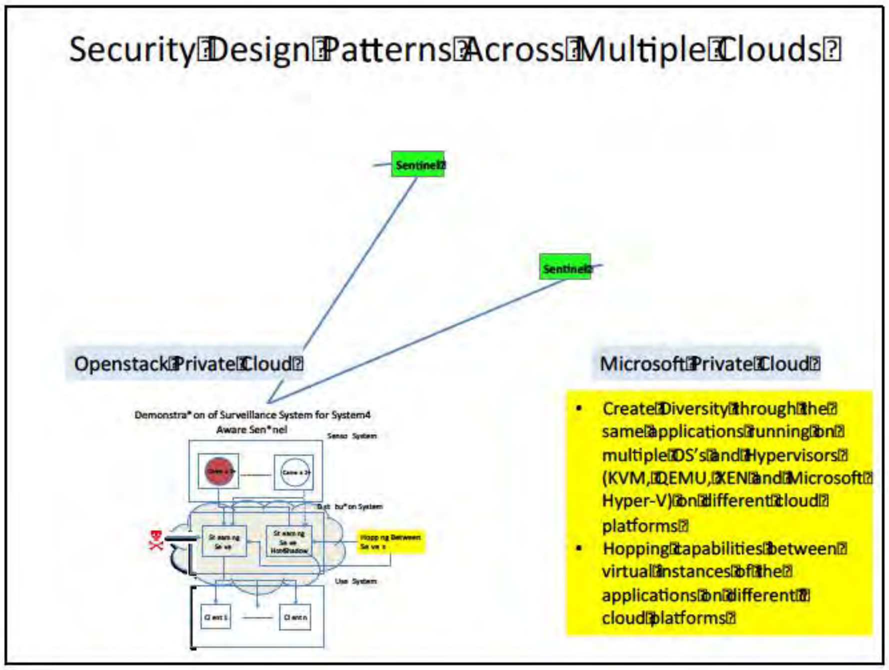


Figure 3 - Demonstration Security Video Distribution System



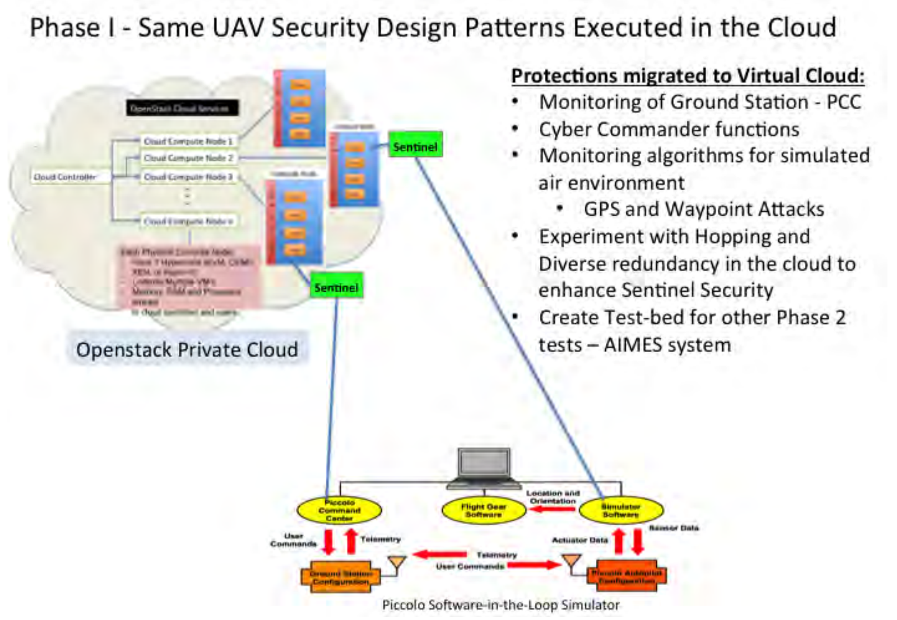
Security Design Patterns Across Multiple Clouds

- Create Diversity through the same applications running on multiple OS's and Hypervisors (KVM, QEMU, XEN and Microsoft Hyper-V) on different cloud platforms
- Hopping capabilities between virtual instances of the applications on different cloud platforms

Figure 4 - Using System-Aware Design Patterns to Protect and Application

2.5.1.2. Use-Case: The UAV Sentinel Applications in the Cloud

Though the cloud computer environment lacks the physical interfaces necessary to hook directly to the physical Piccolo II autopilot, system we were able to utilize the software-in-the-loop (SiL) simulation environment provided by Cloud Cap Technologies in order to emulate the aircraft in flight and to generate attacks on parts of the flight system. We were then able to successfully implement the same protections we developed for use on a physical Sentinel protecting our real UAV system with the same code base on to virtual computing platforms running inside the Openstack Cloud-based infrastructure. This enabled a test-bed for trying diversity and hopping security design patterns on applications and gave us a base-line for comparisons for other applications that could be protected using the private cloud as a hosting platform. Results from this process indicated that the timing and latency impacts on the Sentinel protection algorithms run in the private cloud versus running in the physical instantiation of the Sentinel on single board computers (SBC's) much like the Raspberry Pi's used in the UAV System-Aware example, yielded similar good results. This provided us with a simple way to alter the operating conditions for the Sentinel logic in terms of resources (size), operating systems and host hypervisors, and provided a mechanism to scale the supporting Sentinel architecture on an as-needed basis to either larger or smaller platforms. These results encouraged us to move forward to the next step in the research process, which was to investigate and to implement a more real-world test-bed application, in this case, the AIMES video exploitation system from Leidos.


Phase I - Same UAV Security Design Patterns Executed in the Cloud

Protections migrated to Virtual Cloud:
- Monitoring of Ground Station - PCC
- Cyber Commander functions
- Monitoring algorithms for simulated air environment
  - GPS and Waypoint Attacks
- Experiment with Hopping and Diverse redundancy in the cloud to enhance Sentinel Security
- Create Test-bed for other Phase 2 tests – AIMES system

Figure 5 - Cloud Implementation of System-Aware / Sentinel Protections on the UAV system functions

2.5.2. Issues in Test-bed Configuration

Initial attempts at configuring the cluster to work with the lab's private network infrastructure proved to be a challenge. The scripts used to install the Openstack software made use of DNSMASQ, a lightweight DHCP server. This server conflicted with the DHCP currently established in the lab's network, leading to an incorrect allocation of IP addresses and ranges which control the distribution of address to the virtual computing environments hosted within the Openstack cloud architecture. After eliminating the problem by disabling DNSMASQ, we successfully instantiated virtual machines (VMs). Each VM has direct access to the Internet through an assigned public floating IP address from a pol of addresses made available to this effort. Through this Internet connection, we pulled all sentinel functions from the project team GitHub repository which maintains the development code-base for System-Aware protections and for the open source libraries which are needed to support the monitoring and detection functions of the Sentinel logic.

Our first VMs functioned only as headless servers with no user interface. We desired the ability to run the cyber commander GUI interface (the main Sentinel alerting and correction interface for the UAV example) in a cloud environment, however, necessitating an exploration of VNC in OpenStack. The administrative view of the OpenStack dashboard provides basic VNC functionality. Exploration of the OpenStack Nova API revealed commands that return a URL which allows http access to virtual machines using a standard web browser. After instantiating a Ubuntu 14.04 desktop VM, we successfully connected to the controller node using an SSH connection on the university's public network. With the returned URL, we accessed the VM with no apparent degradation in speed or responsiveness.

2.5.3. Creating the AIMES Test-bed Environment

As a second use case, we have successfully implemented a working demonstration system of the (Advanced Imagery Exploitation System - AIMES) in order to experiment and test methods for securely protecting ground-based systems that are directly interacting with an airborne system. The ground-based system will employ a private Cloud-based Sentinel that provides agility features that can help to protect the Sentinel from attack. Our intention is to evaluate the agility limits of diverse redundancy and configuration hopping solutions, and also the complexity trade-offs of voting algorithms for cloud-based monitoring of a real-world system. As the research continue to progress, the experimentation will be based upon results obtained through the monitoring of a COTS-based UAV imagery processing system produced by Leidos Corporation. Leidos has provided a software license to UVA for AIMES, and support for integrating AIMES into the UVA simulation laboratory and will provide support for determining the necessary information required to protect AIMES, and for supporting testing of the possible AIMES system responses to simulated attacks and corresponding Sentinel inputs to AIMES. The activities achieved during this portion of the effort have included 1) Integrating AIMES into a UVA simulation evaluation environment which has been a, 2) Designing a working cloud-based Sentinel prototype on simulated system. The efforts on the AIMES environment continue to address conducting simulation experiments that allow for trade-off analyses of Sentinel complexity vs. attack detection and design evaluations of Sentinel security vs. Sentinel complexity, using trade-off results as a basis for developing a Sentinel design methodology for cloud-based Sentinels, and identifying potential initial operational prototype opportunities for advancing the work beyond this AIMES-based effort.

3 AIMES System– An Example of Applications in the Cloud and As a System to Apply System-Aware Techniques

3.1. AIMES Overview

AIMES (Advanced Imagery Exploitation System) is a video exploitation system from Leidos that is used by the Air Force and the Army to provide intelligence capabilities on video streams coming from video surveillance platforms in the field. The three central VPC AIMES components include the primary image capture and transformation element (PICTE), the AIMES motion imagery library, and the desktop exploitation software (AIMES Exploit).

The image capture element receives all real-time, full motion video (FMV) data, synchronizes the video and embedded information (such as time and geolocation), and transforms the synchronized multiplexed stream into a standards-compliant output stream that is used by the rest of the AIMES system. The streaming output from the PICTE can be viewed immediately in real-time using the AIMES exploitation software, or archived in the motion imagery library. The AIMES motion imagery library server is the storage processing system, which is used to store the streams from the PICTE, as well as serving the video to the AIMES Desktop Exploitation Client, which provides analysts access to the video and to the system user functions necessary to tag and analyze the video data.

The AIMES system supports the ingestion, tagging, and viewing of MISP compliant full motion video streams. The AIMES server connects to sensor platforms and receives data in real time, streamlining an analyst's job of communicating findings and generating intelligence reports. Ingestion of the FMV data involves storing both slice and frame information in a PostgreSQL database. Each video frame is associated with metadata describing the position and state of the sensor platform, such as latitude, longitude, and altitude.

3.2. The AIMES Test-bed Implementation

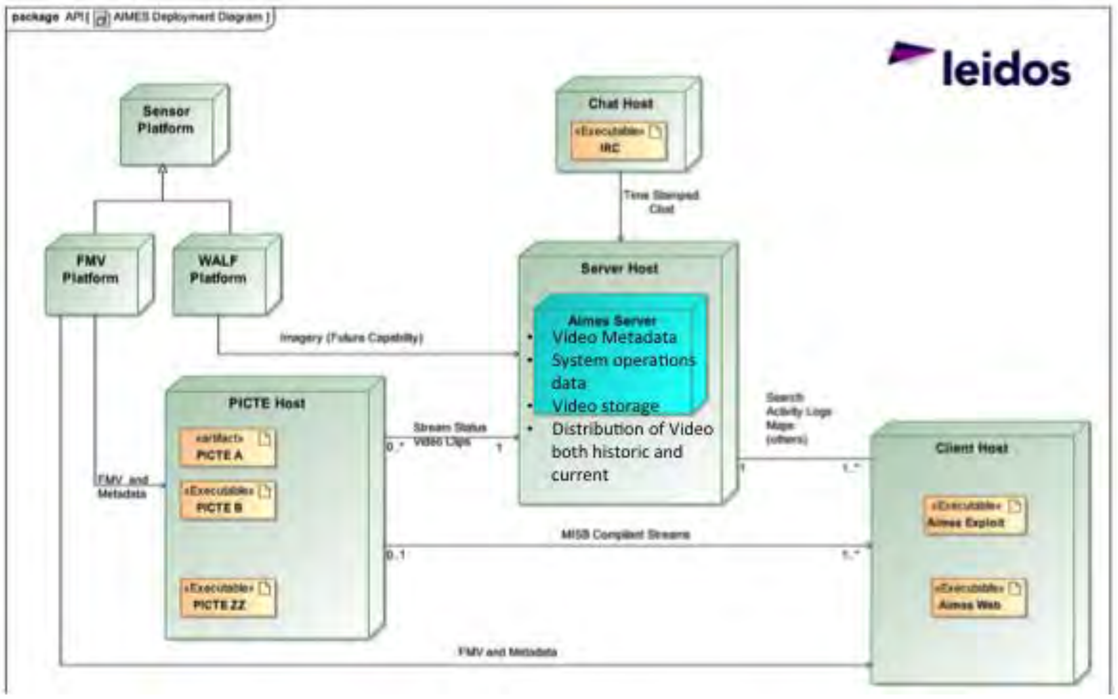


Figure 6 - Overview of AIMES System Architecture

The AIMES system consists of four primary components: the sensor platform, the PICTE server, the ADMST server, and the AIMES Exploit software. For purposes of developing both attacks and System-Aware Cybersecurity / Sentinel protections, the current configuration does not have direct access to an actual sensor platform. Instead, MIRestream software is used to multicast video data from a predetermined location in the file system, simulating the existence of a sensor platform. The PICTE server receives the FMV feed from the sensor platform and gives the feed a unique identifier. PICTE provides direct control over the source and destination of the FMV stream, either through unicasting or multicasting. The stream information is received by the ADMST server, which marks the current feed as active in the PostgreSQL database. Additionally, PICTE handles the transfer of the video files using FTP to the server. The server is responsible for monitoring an ingest directory and extracting all video metadata for storage in the database. Additionally, a set of web services provided by the server, allow client machines running the AIMES Exploit software to send requests in a Simple Object Access Protocol (SOAP) format. The server responds by providing the information necessary for the client to find and view the requested streams. Additionally, analysts can perform searches using a variety of other parameters such the contents of a chat message log associated with a particular video stream.

3.3. Virtualization for the AIMES Application Test-bed

The current configuration of the system allows for a smooth transition to a fully virtualized environment. MIRestream and PICTE each run on a separate virtual machine running Windows Server 2008. These virtual machines are managed using the Hyper-V hypervisor on a physical machine running Windows Server 2012. The ADMST server and the AIMES Exploit client software both run on physical hardware – though the ADMST server is perfectly viable for a virtual delivery. The client software, AIMES Exploit, requires updated graphics drivers that are not compatible with a virtual environment. We plan to virtualize the ADMST server software as the project research progresses. All system components are configured to communicate over a shared private network. All required data for streaming is stored directly in the file systems of the physical and virtual machines, so no public network interface is required. As we move towards evaluating the application in the private cloud environment and applying System-Aware techniques to protect both the application data streams and the integrity of the cloud infrastructure which supports the Openstack private cloud, we will migrate the virtual environment into the cloud.

3.4. Progression of Test-bed Configurations

Initial implementations of the test-bed involved minimal hardware so we could focus our efforts on understanding the installation options and the functions of components that make up a working AIMES implementation. This was accomplished with the help of Leidos system designers. The first version of the demo system was configured to entirely on one machine, including the client software. This enabled the team to become familiar with the installation of the system.

Our second implementation occurred on two machines with the client software extracted out to a separate laptop. This setup was used for quick and relatively portable demonstrations of the system's capabilities while beginning the process of understanding the distribution of system functions and the mechanisms for video dissemination from the sources to the analysts performing user functions.

In our last implementation iteration, we separated the individual system components with the intent of accurately mirroring how the system is actually deployed, and to enable our ability to insert ourselves between the system components in order to simulate attack that might emanate from compromised insiders or supply chain-based attacks. In an attempt to leverage access to powerful physical hardware, we attempted to freshly install the components in a Windows Server 2012 environment. However, the supported platform specification for AIMES is Microsoft Windows Server 2008. Configuration of the PostgreSQL database in Server 2012 resulted in multiple fatal system errors. After troubleshooting efforts, the project team elected to virtualize the Windows Server 2008 environment rather than replace the existing operating systems on the physical hardware.

Once the virtual machines had been instantiated and the Windows 2008 operating system installed, installation of the individual components proceeded smoothly, with most of the detailed configuration handled by installation scripts provided by the Leidos team. MIRestream and PICTE were installed on two virtual machines and initial attempts at running the system showed that the two components were successfully communicating. MIRestream multicasted the raw video data, and the PICTE received and processed the video into sixty-second slices, as the system dictates. A link between these components and the server remained missing, however, with the active feed not appearing in the database. We remedied this problem by changing the target address of the server's metadata broker service in PICTE to the actual IP address of the server machine. Before, the service was assumed to run on local loopback, meaning that all stream information was isolated from the server machine. With the target address fixed, the server saved the active feed identifier into the database, and the client could query the server for active feeds and view the live-streamed video.

Upon examining the contents of the database, however, we realized that the server was not storing any video metadata in the database. In fact, it was not receiving any video data at all. After reading through documentation for the server, we realized that the only remaining missing component was a simple FTP server running on the same machine as the ADMST server. Once this was installed and the appropriate credentials were provided to PICTE, video data was successfully transferred to the server machine and ingested into the database.

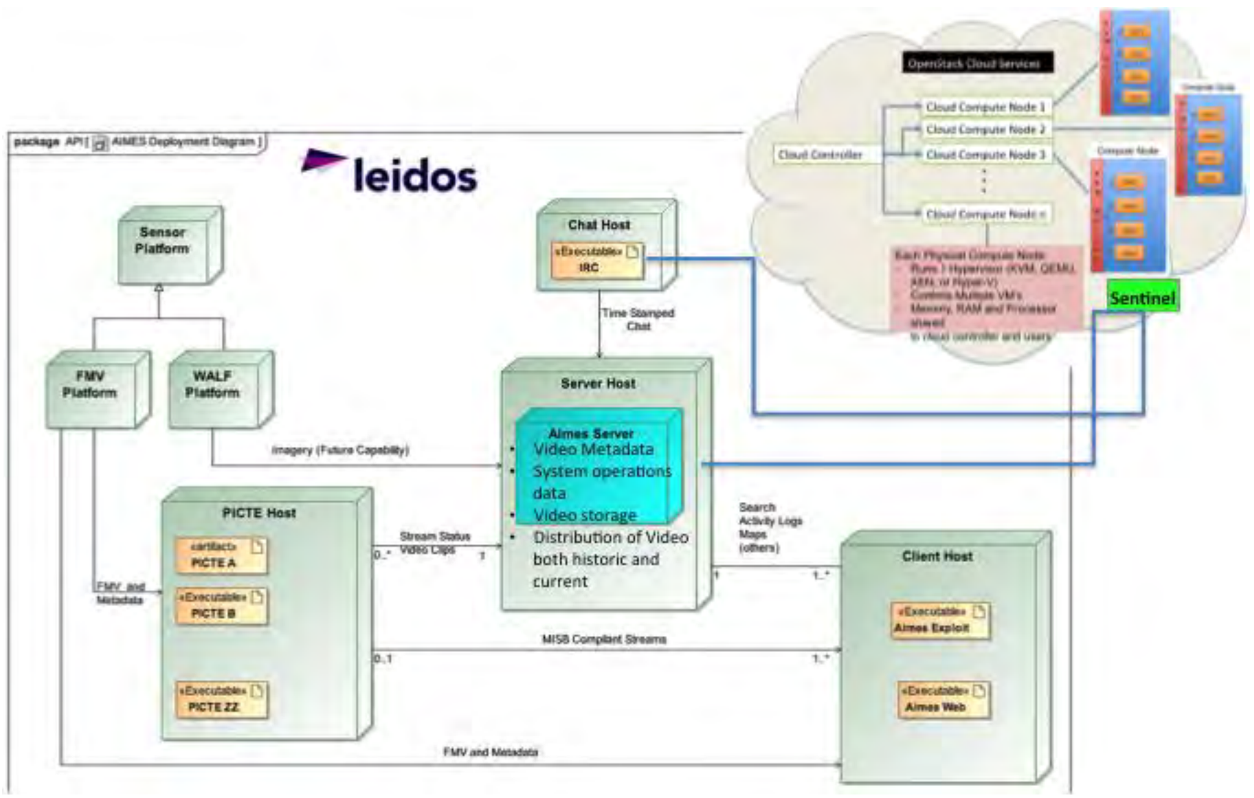
3.5. Potential Attack Opportunities

Once all components had been completely separated, we focused our attention on identifying potential vulnerabilities and attack vectors. In parallel, we also consider possible monitoring mechanisms to address these attacks. Currently, we are pursuing two possible points of attack.

- The first involves tampering with or rerouting the raw video stream, possibly introducing replays or large latencies into the video monitoring in order to obfuscate illicit activities.
- The second involves using SQL injection to taint the contents of the PostgreSQL database, which would render the forensic analysis capabilities of the AIMES system inaccurate and unreliable.

In configuring the system, we recognized that no ingestion to the database occurs if the raw video files are not transferred to the server by PICTE. This characteristic is ideal from the perspective of an attacker; however, as the FMV feed still streams successfully to the AIMES exploit software on the client. This suggests that the client machine only needs to know the address of the stream to view it. The client software does not perform any check against the information stored in the database as video is streamed to the client. An analyst can still perform search queries and receive access to active feeds, with the only degeneration in aesthetics being the lack of a thumbnail in the feed description. This results because no links to extracted thumbnails have been saved in the database. A manually saved thumbnail could quickly mask this visual quirk. Regardless, the direct streaming of video to the client invites a man-in-the-middle attack, which fundamentally alters the video received by the client and will be the primary focus of our initial attack approach.

As a secondary attack avenue, we have evaluated injecting our own commands into the database, basically an SQL injection attack, and have deemed it as a viable, though somewhat less promising attack vector. Metadata saved to the database is packaged in an XML file as part of an HTTP request sent to the server. Initial searches show that the mechanism by which the server executes the commands to save the video metadata is hidden inside compiled code. The database could play a larger role in the detection of attacks; by comparing the stored metadata with the metadata of the incoming video stream, the current state of the incoming stream may be validated. Our intention are not to affect changes on the source code itself, but to rely on attack components that either insert attacks between components or rely on attacks on the cloud infrastructure to modify system behaviors and then to enable the System-Aware protections in order to protect those system functions.



3.6. Ensuring the Private Cloud Infrastructure

A comprehensive effort is required to ensure the integrity of private cloud infrastructures. Clouds, like other IT infrastructures, are certainly vulnerable to cyber security attacks. If the infrastructure of the cloud is compromised, then the applications supported by that infrastructure are also vulnerable. In this part of the effort, we looked to investigate the feasibility of creating a theoretical security monitoring framework for assuring cloud infrastructure integrity and providing example implementations as a step in proving feasibility. We identified a sample attack, designed and implemented the infrastructure to monitor for such attacks, and designed and implemented classes of techniques to respond to such an attack.

In considering a new security monitoring framework for cloud architectures, we must first investigate the potential risks inherent in the infrastructure itself and potential threats to the data that moves in and out of the systems hosted on the cloud platform: Disrupting Cloud , Data Barriers Within the Cloud, Data Barriers at the Cloud , Data Confidentiality and Data corruption. These are described in detail in Section 2.2.1. Areas that offer potential attack vectors in the Openstack platform are highlighted in Figure 7.

To ensure that our research is manageable, relevant, and scoped properly, we began our FY14 efforts by focusing on cyber attacks that successfully introduce *application latencies* within a private cloud infrastructure -- that is, cloud applications continued to operate, except at a pace slower than designed and/or expected. For example, applications would deliver information to each other later than "normal", prevented via a number of attack mechanisms within the cloud networking layer, or within the cloud infrastructure services, or within the cloud infrastructure software, etc.

To develop a theory for correct private cloud operations, we decided to pursue a *bottom-up* methodology, in which a specific cloud deployment provides the concrete basis for research. That is, we decided to construct a plausible and narrow cyber attack on a specific cloud with corresponding specific cyber attack discovery mechanisms and mitigation strategies. We chose the most popular and important open-source private cloud software, OpenStack. We pursued a research plan of gradually expanding the scope of cyber attacks on OpenStack clouds, with the intent to generalize in all dimensions as the research progressed.

We further divided our increased-latency cyber attack into two equivalence classes: operations necessary for correct *cloud application* behavior, and operations necessary for correct *cloud* behavior. In the first category, we focused on the cloud object store ("*OpenStack Swift*"), and in the second category, we focused on cloud scheduler operations ("*OpenStack Nova*"). While we made progress on bother equivalence classes, our development efforts focused on cloud scheduler operations.

We studied approaches to ensure that scheduling of virtual machines (VMs) was reasonable in a private cloud that might be under attack. For example, a successful cyber attack could result in new cloud activities (VMs) be placed incorrectly on already-heavily-loaded physical machines. Normally, simplistically, VMs should be placed onto physical machines that are relatively lightly loaded. A successful attack would then significantly slow down the operation of the newly-spawned cloud application (VM) as it contends for resources already under duress.

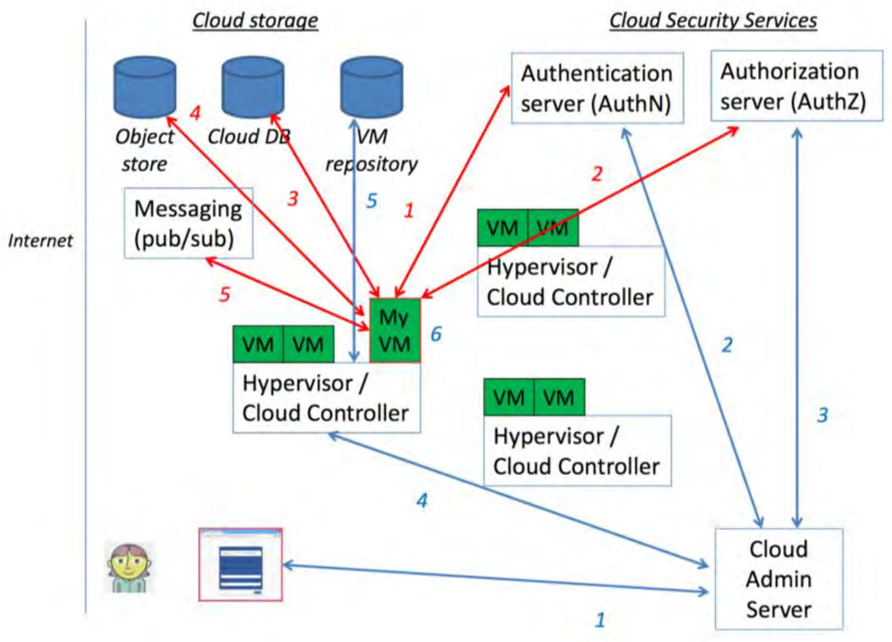


Figure 7 - Diagram of possible attack vectors within the Openstack Infrastructure

After studying the OpenStack design and software, we determined that there were two best options by which to detect this cyber attack:

- ***Cloud Infrastructure Reporting.*** We modified the OpenStack software by adding event-driven notifications both (a) to measure the duration for a new VM to be scheduled and report into the cloud monitoring infrastructure; and (b) to measure the duration of cloud object storage operations ("blob store"). The basic idea was these notifications would provide experimental timing information as the baseline for normal (correct) private clouds operations; these baseline measurements would then be consulted as the basis for asserting anomalous behavior. Reporting of scheduling duration did not require any new methods be added to OpenStack. We did need to modify existing methods to add timing information and then we needed to invoke the existing notification sub system. Adding notifications to the blob storage system was a little more involved. OpenStack is currently transitioning between notification systems and while the scheduler uses the newer notification system the blob storage system still uses the older one. After trying to hook into the older system we ended up adding code to the blob storage system that would allow us to send notifications through the newer notification system. In assessing the results of the new notifications we realized that if the cloud was compromised and the scheduler was getting bad information it would make bad decisions in about the same time that an uncompromised system would make good decisions. So while the scheduler notifications would detect an attack on the node that was responsible for making the decisions it wasn't very effective at detecting when worker nodes were compromised and sending bad state information to the scheduler.

- ***Active Probing.*** In contrast to the first approach, in which the pre-existing cloud monitoring system was modified to collect new information (and use that information as the basis for asserting that a cyber attack is occurring), we also designed a self-contained, new activity that was scheduled on-demand and whose functionality was to actively probe the infrastructure for anomalous behavior. Generally, this is the basis of a self-contained infrastructure that can be scheduled and executed *outside* the OpenStack mechanisms (such as VM scheduling). In this case, this active probe functionality attempts to perform an independent assessment of the computational load on the physical machine, and then use probe-specific mechanisms to report its results to the probe-initiator.

We believe that diverse mechanisms provide a more reliable mechanism by which to detect the anomalous behavior, and thus provide complementary detection information. Each approach has different pros and cons. The advantage of "Cloud Infrastructure Reporting" is that it is passive and measures how long it takes for work that needs to happen anyway. It adds very little overhead to the existing system. However, there are some natural variances in how long operations take on an active node so finding an appropriate threshold was challenging. Too low and false positives will quarantine healthy nodes while too high will delay detection of an attack. The "Active Probing" approach allows us to run some code to measure the health of the machine. In our prototype code, we opted to run a quick benchmark on the system. With the right benchmark you can also probe individual sub components of a node so you can gain more information about why a node might be slow. The disadvantage of this approach is that time spent running the benchmark is time that could be spent doing some more useful computation in a VM. If the benchmark is run too often or runs for too long you will achieve the goal of slowing down the system for the attacker. To ensure comprehensive results, we chose the Pystone benchmark. While Pystone doesn't give performance numbers for system components, it runs fairly quickly and is easily configured to run for different durations. Also while we only investigated a homogeneous cloud we believe that it will be easier to reason about node performance in a heterogeneous cloud with benchmark numbers that just the durations of the passive approach.

In the event that either of these cyber attack mechanisms are triggered and assert that a latency-introducing cyber attack is being executed successfully on the private cloud infrastructure, we designed and implemented a mechanism by which to identify those nodes suspected of being compromised. This was also performed via a new, probing mechanism. These nodes were subsequently removed from future scheduling decisions. We are also researching approaches by which to further isolate those suspected nodes from the overall continuing operation of the private cloud.

4 Future Work

The next steps for this part of the research program are to generalize various aspects of the particular attack and response (introduce new cyber attacks that introduce latencies in cloud operations, expand and generalize mechanisms by which to discover successful attacks that introduce latencies, and expand and generalize attack mitigation strategies). While the focus on OpenStack, we argue, is an important means by which to create concrete and relevant results (Openstack is the most widely-deployed open source software for private clouds), we recognize that an important future step is to move beyond the particular software architecture and philosophy of OpenStack and generalize to other private clouds. Additionally, we have identified and are currently pursuing threats beyond latency. For example, we are beginning to expand the general functionality of the probe to more properly regulate frequency of probe, intent of probe, better handling of false positives, better handing of false negatives, etc.) We are also creating a theory and mechanism beyond the probing mechanism (e.g., through the widespread use of cryptography such as via digital signatures).

Research areas scheduled for activities in Phase 2 of the cloud activities include:

- Further developing the prototype AIMES cybersecurity sentinel. The researchers shall identify attacks that are of concern to operators, which can be detected and possibly contained through use of the Sentinel. The researchers shall investigate using diverse redundancy in terms of multiple cloud implementations, perhaps geographically separated as well, for monitoring and reconfiguration.
- Develop and conduct experiments on a diverse set of integrity assurance functions for use while the cloud-based Sentinel is performing its operational functions. Such functions should include active testing of cloud performance. The SERC researchers should investigate the potential to discover situations where system parameters, data storage locations and data distribution methods could be exploited.

The activity focused on the monitoring of a Cloud for integrity confirmation produced the early stages of a concept for actively testing a Cloud through a Sentinel by initiating test SW programs for execution and observing Cloud responses as described above. In particular, this concept was applied in a test case for determining if an adversary had forced a Cloud platform to bypass its normal methods of deploying a virtual machine. Specifically, we looked at a hypothetical case where an adversary forced a virtual machine to be deployed to an overloaded piece of hardware in order to introduce latencies into the functions on that virtual environment.. This cybersecurity design pattern of test during operational use of the Cloud can potentially be expanded to discover situations where adversaries manipulate system data values, misdirect distribution of data, modify data storage locations, etc.

Based on these results, this project will advance these initial findings by:

1. Developing a prototype of an AIMES cybersecurity Sentinel that is employed on a Cloud. This will entail identifying attacks that are of concern to operators and that can be detected and possibly contained through employment of the Sentinel. In addition, as part of assuring the operation of the Sentinel, we will explore employment of diverse redundancy in terms of multiple Clouds (perhaps geographically separated) for monitoring and reconfiguration. Prototype results will be made visible to the AIMES user community to gain interest in starting to develop a transition strategy for the Cloud-based Sentinel concept.

   The ongoing FY14 project has started the learning activity regarding AIMES that will be required to select and implement attacks to defend against and to determine monitoring opportunities to support attack detection. The needed UVA learning efforts have been and will continue to be supported through a Leidos software support effort that provides UVA with needed information related to potential cyber-attacks and cybersecurity pertinent to the design of a prototype AIMES Sentinel (AIMES is a Leidos-developed software system).

2. Expanding the active testing concept described above to include the development of and experimentation on a diverse set of integrity assurance functions and operate those protections while the Cloud is performing its operational Sentinel functions. Two example Cloud infrastructures will provide the basis for gaining experimental results regarding the boundaries of applicability of the active testing concept and the performance within those boundaries. The results of this effort will potentially define a pathway for advancing Cloud security beyond the application to Sentinels that will be exposed to the wide range of DoD stakeholders that are interested in employing private Cloud-based systems.

SYSTEMS ENGINEERING
Research Center

Security Engineering Project

Part 4 – Preliminary Investigation on the Integration of SysML Models and Attack Tree Models for Productive Cyber-Vulnerability Analysis and Selection of Cybersecurity Solutions

Principal Investigator: Dr. Barry Horowitz, University of Virginia

Co-PI: Dr. Peter Beling, Associate Professor, Systems Engineering, University of Virginia

Senior Researchers:

Carl R. Elks
Department of ECE
Virginia Commonwealth University
Richmond, VA

Nicholas R. Bollweg
Georgia Tech Research Institute
Atlanta, GA

Technical Report SERC-2015-TR-036-4 Part 1a

January 31, 2015

Executive Summary

The System-Aware Architectural Selection and Assessment methodology is a process that has been developed as part Phase 1 and Phase 2 of this project in order to identify the critical system components for a particular system, to: 1) identify the possible attack paths to attack those components, to determine which of those attack paths would be most desirable to an adversary, 2) identify possible cyber security defenses against those attacks as well as to evaluate the impacts of those defenses on the attacker, 3) assess the effects on system performance of potential defenses, and 4) estimate the security trade-offs across alternative architectural solutions. The System-Aware Architectural Selection and Assessment methodology is composed of six steps; each step having a well-defined goal, required deliverables, and responsible analysis team(s) for that stage. Phase 1 focuses on a manual approach to implementing these steps. The manual processes developed in Phase 1 include (i) a scoring system that highlights cost-benefit tradeoffs for decision makers and (ii) a method based on influence diagrams designed to aid decision makers in understanding ways in which uncertainty could be introduced into an attacker's outcomes. Phase 1 methodology was successfully applied to the UAV surveillance system, both to develop the concepts through case studies and as a practical method for selection of research targets for the attack and defense teams. From these applications, it became apparent that while the methods left participants with a clear sense of having successfully explored the design space, a set of analysis support tools could make an important contribution for enhancing the manual effort as part of addressing more complex systems as well as mission-based, integrated system-of-system security assessments.

In Phase 2 we began to focus on developing methods for computer-based support to the processes for evaluation of the architectural selections of critical system functions to be protected on a cyber-physical system. The Phase 2 approach is based on exploiting and supplementing existing modeling paradigms with computer-based support based upon open source or commercial software. Specifically, the System-Aware Architectural Selection and Assessment methodology has been recast in terms of use of model-based systems engineering tools, such as SysML, and attack tree tools, such as the commercial package SecurItree. Much of the Phase 2 effort was directed at developing concepts and software to integrate systems models with attack trees to facilitate an iterative architectural design process in which decision makers could easily switch back and forth between defender and attacker views of the system or mission as part of selecting solutions for defense.

The Phase 2 activity culminated in a workshop in November 2014 with participants from 10th Fleet Cyber Command, the Office of the Secretary of Defense the Johns Hopkins Applied Physics Lab and the Center for Naval Analysis. The goal of the workshop was for participants to discuss the complexities of the decisions and tradeoffs inherent in choosing a defensive architecture as well as to introduce the methodology and toolset being designed in this research activity. The format was interactive; participants engaged in the design of a defensive architecture for aspects of the UAV system, including the video surveillance mission. Highlights of the exercise included the use of the architectural scoring tools and an introduction to SysML and cyber-attack tree support tools, such as SecurITree, that can represent the system from an integrated defender and attacker perspective. Discussion focused on opportunities for systems-aware cybersecurity deployment and how future versions of the tools might best support decision makers. The workshop provided useful perspectives regarding the future opportunities for enhancing and using computer-based support tools that served to impact future research plans.

Table of Contents

1. Introduction 227
2. System-Aware Architectural Selection Framework 229
  2.1. Architectural Selection Methodology 230
  2.2. Model-based Approach to Architectural Selection 232
    2.2.1. SysML for System-Aware: Concept of Employment 232
    2.2.2. SysML4SystemAware 233
    2.2.3. Overview of Attack Trees 233
      2.2.3.1. Prerequisites of an Attack 236
      2.2.3.2. Modeling Threat Actors 236
      2.2.3.3. Modeling Victim Impacts 238
      2.2.3.4. Issues with Static Attack Tree Methods 238
    2.2.4. Overview of the Systems Modeling Language 239
  2.3. Integration Principles 239
    2.3.1. Authoring tools 239
    2.3.2. Attack Tree Authoring 240
    2.3.3. SysML Authoring 240
    2.3.4. Data Types 240
    2.3.5. Attack Tree Modeling Language 240
    2.3.6. SysML XMI 241
  2.4. Data and Model Consistency 241
  2.5. Visualization 241
    2.5.1. Configuration Management 242
    2.5.2. Integration Environment 242
3. Architectural Assessment Workbench Concept 242
  3.1. Initiate System Model 243
  3.2. Visualize System Model 243
  3.3. Identify Exploits 243
  3.4. Model Exploits 244
  3.5. Model Defenses 244
  3.6. Compare Exploits and Defenses 244

3.7. Present to Stakeholders 244

4. Overview of the Baseline System: UAV autopilot and Camera System 244

5. Case Study: A Baseline System Model for a Notional Intelligence, Surveillance and Reconnaissance Mission 246

6. Case Study: Modeling the Effects of a GPS Attack 252

7. Case Study: Modeling Defense of a Mission 254

8. Case Study of SysML Informed Attack Tree: UAV Walk-off Attack 255

9. Attack Tree Modeling Process with SysML Models 258

10. Querying Models to Explore Potential Vulnerability Space 259

11. Formulating Path Graphs from Attack Tree Dashboard 259

12. Findings and Lessons learned 264

13. Planned Activities for 2015 264

14. References 266

List of Figures

Figure 1: Spectrum of approaches to vulnerability analysis. 228
Figure 2 - Levels of cyber vulnerability analysis. 229
Figure 3 - Simple Example Attack Tree to Demonstrate Notation 234
Figure 4 - Utility Functions for Agent Profiles 237
Figure 5 - Architectural Assessment Workbench Concept 243
Figure 6 - General Architecture of the Autopilot 246
Figure 7 - SysML Requirements Diagram of ISR Mission 247
Figure 8 - SysML Profile Diagram of SysML4SystemAware Profile 247
Figure 9 - SysML Block Definition Diagram of ISR Mission 248
Figure 10 - SysML Internal Block Diagram of the ISR Mission 248
Figure 11 - SysML Package Diagram with Allocation of Mission Activities 249
Figure 12 - SysML Block Definition Diagram of Information Hierarchy 249
Figure 13 - SysML Block Definition of Platform Component 250
Figure 14 - SysML Internal Block Diagram of Platform Subcomponent Connectivity 251
Figure 15 - SysML Block Definition Diagram of Platform's Autopilot Structure 252
Figure 16 - SysML Internal Block Diagram of Autopilot 252
Figure 17 - SysML Block Definition Diagram of an attacked ISR Mission 253
Figure 18 - SysML Block Diagram of a More Detailed understanding of the ISR Mission 254
Figure 19 - SysML Block Definition Diagram of a System-Aware defended ISR Mission 254
Figure 20 - SysML Requirements Diagram of a System-Aware defended ISR Mission 255
Figure 21 - SysML Model for the Autopilot 257
Figure 22 - Attack Tree Creation Process without SysML 258
Figure 23 - Relationship Diagrams in the Autopilot 259
Figure 24 - Query Seach Results 260
Figure 25 - Block Diagram of a Typical AdaFruit GPS Receiver 261
Figure 26 - GPS Code Snippet 262
Figure 27 - Attack Tree with New Path Inserted 262
Figure 28 - Attack Tree with Attack Profiles 263

List of Tables

Table 1 - Stereotypes in the SysML4SytemAware profile. 233
Table 2 - Results of Attack Pathway Calculations.. 264

1 Introduction

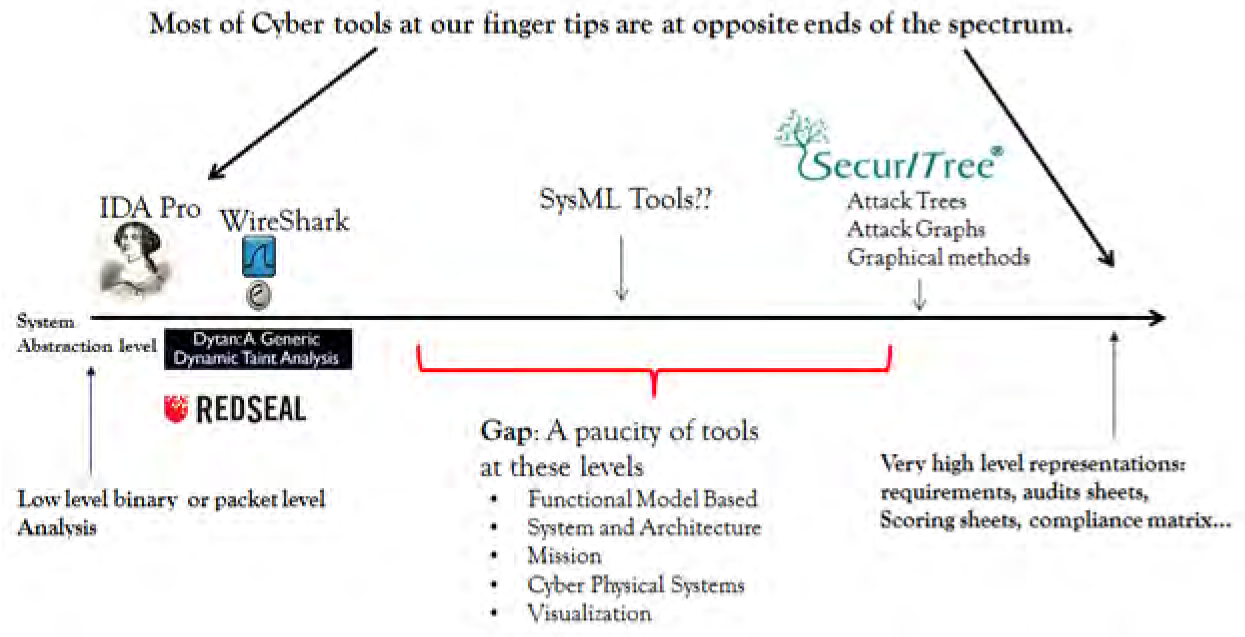
At present, the state of the practice for augmenting Cyber Physical Systems security is primarily perimeter-based security (such as firewalls, intrusion detection mechanisms, anti-viral signature software, encryption, and advanced user authentication). However, the trend in adversarial attacks is moving toward well-formed coordinated multi-vector attacks that compromise systems in such a way that detection and identification of attacks is challenging through perimeter security solutions and human monitoring alone [Wulf and Jones 2009]. These styles of attacks are possible due to the recent advent of stealthy Advanced Persistent Attacks (ATP's) - where attackers penetrate firewalls and intrusion network devices by very sophisticated malware, and then quickly steal information on the operational aspects of critical infrastructure systems before they are detected, if they are detected at all [[i]]. The information retrieved by an adversary from a "smash and grab" ATP attack is subsequently used to craft special multi-vector malware that targets the critical digital components of infrastructure systems.

As an end result, it has been recognized that perimeter security needs to be strongly augmented by other approaches for addressing the current and emerging cyber threat potential for Cyber Physical Systems [[ii], [iii], [iv]]. Reliance and dependability of CPS will depend upon not only on the understanding of threats, but also upon the development of well-engineered science based approaches to protect critical assets that; (1) are effective at deflecting multi-vector APT attacks, (2) do not interfere with safety, dependability and reliability aspects of the CPS, and (3) reverse the cyber asymmetry from the attackers' advantage to the defenders' advantage.

Vulnerability analysis today is performed largely at two extremes as shown in Figure 1; at the very low level which is typified by binary analysis and debugging and use of packet analysis tools, and conversely at the very high level which includes tools like attack trees, graphical methods, and tabular auditing methods. At the low level, the focus is on very specific interactions at the program machine interface level, which largely excludes larger system context. At the higher level, the focus is on system level with focus on postulated attacks or vulnerabilities with respect to the system, with little support to determine what is the real set of vulnerabilities and possible exploits of those vulnerabilities. Consequently, today we practice Cyber-systems vulnerability analysis in a disjointed, fragmented fashion

[i] F. Yarkochkin, "Hunting in the Shadows: In depth Analysis of Escalated APT Attacks", BlackHat Conference 2013, Las Vegas, Nevada
[ii] NIST 2013 Workshop on the Foundations of Cyber Physical Systems, prepared by Energetics Corporation. http://www.nist.gov/cps/
[iii] A.A. Cardenas, T. Roosta, and S. Sastry. "Rethinking security properties, threat models, and the design space in sensor networks: A case study in SCADA systems". Ad Hoc Networks, 2009.
[iv] GAO. Critical infrastructure protection. Multiple efforts to secure control systems are under way, but challenges remain. Technical Report GAO-07-1036, Report to Congressional Requesters, September 2007.

that leaves recognition of what can be important exploits undetected, due to the fact that a systems perspective of cyber-vulnerability analysis is lacking. We argue that a model base systems perspective can provide the basis for a potential solution to this dilemma, guided by well-formed principles of architectural level cyber analysis. This would bring the power of system level modeling to enable a more holistic perspective on cyber security.



Most of Cyber tools at our finger tips are at opposite ends of the spectrum.
IDA Pro
WireShark
SysML Tools??
SecurITree
Attack Trees
Attack Graphs
Graphical methods
System Abstraction level
Dytan: A Generic Dynamic Taint Analysis
REDSEAL
Gap: A paucity of tools at these levels
Functional Model Based
System and Architecture
Mission
Cyber Physical Systems
Visualization
Low level binary or packet level Analysis
Very high level representations: requirements, audits sheets, Scoring sheets, compliance matrix...

Figure 1: Spectrum of approaches to vulnerability analysis.

The key advantage of employing a model based cyber-vulnerability analysis approach is viewing the system from several important perspectives and dimensions of abstraction within a framework. These perspectives include the mission perspective, which provides context to the consequences of exploiting vulnerabilities, the architecture domain which embodies the organization and relationships between sub-systems and platforms, and the platform domain, which represents all of the functions, and components that are working together to achieve a mission. Figure 2 shows the concept of model based CVA.

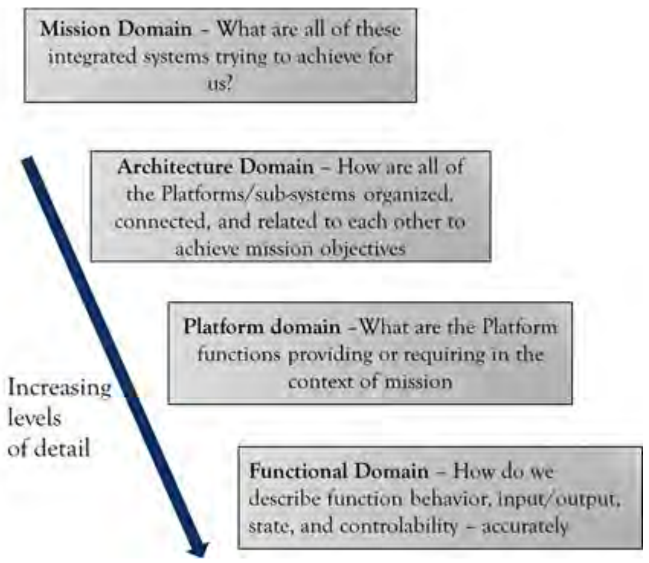

Mission Domain – What are all of these integrated systems trying to achieve for us?
Architecture Domain – How are all of the Platforms/sub-systems organized, connected, and related to each other to achieve mission objectives
Platform domain –What are the Platform functions providing or requiring in the context of mission
Functional Domain – How do we describe function behavior, input/output, state, and controlability – accurately
Increasing levels of detail

Figure 2 - Levels of cyber vulnerability analysis.

Model based analysis allows us to support decision making by providing reasoning along these dimensions. It provides models to collect insight that otherwise could be overlooked. It allows us to integrate information from a variety of relevant vulnerability analysis tools that address the spectrum of Cyber Vulnerability Analysis needs, such as Attack Trees tools, to the framework. It enables us to assess the criticality of platforms and functions with respect to mission, and finally it allows us to evaluate potential cyber-defenses in a broad-based and objective manner, accounting for the multiple classes of attacks and threat agents.

An important question is, why do we want a multi-perspective model of the system? While modeling at this level requires a hopefully modest upfront investment (one of the important objectives of this research effort relates to cost and productivity regarding employment of advanced analysis tools), the model becomes a living representation of the system, allowing system engineers to support decision-making activities that must evaluate the impact of modifications or upgrades to the protected system in the context of preserving security for various missions. Perhaps most importantly, through information extracted from the model via querying tools, decision makers can achieve higher levels of confidence in their selection of cyber security solutions. The model-based approach has the potential to provide insight into important questions such as:

- Did we overlook important attacks or important defenses?
- How do we know if an attack/defense is important?
- If we do choose to add a defense solution, what are collateral impacts on the system to be protected, and how do these impact our integrated defense posture.

2 System-Aware Architectural Selection Framework

A key objective in System-Aware Cybersecurity is to reverse the asymmetric advantage enjoyed by attackers by identifying areas where the defensive team can make minimal changes to the system that will cause a maximum increase in difficulty, uncertainty, or expense for the attacker. A wide variety of design patterns for functional defense have been developed using concepts such as data provenance, moving targets diverse redundancy, voting schemes, and sentinels. Prototype and simulation-based System-Aware Cybersecurity solutions have been evaluated for a number of cyber-physical systems including shipboard control systems, power-generating wind turbines, and video reconnaissance systems on unmanned aerial vehicles (UAVs).

The most basic decision problem in designing a Systems-aware Cybersecurity solution, which we term the architectural selection problem, is that of selecting which system functions to protect and, for each function, choosing a collection of defensive measures from among a rich library of design patterns. To make such choices, the designers of defenses should have deep understanding of the system to be protected, including an understanding of the relative importance of each function in the estimation of the system owners and operators

Finding the best solution is a complex multi-criteria decision analysis problem with several distinguishing characteristics:

- Solutions must account for the actions and capabilities of the adversary, now and in the future.
- Solutions must satisfy a large and diverse group of mission stakeholders.
- A mission solution involves multiple systems, each needing to be secured.
- Solutions for a mission area must account for the particulars of each of the systems being protected and their relationships to mission outcomes.

The complexities associated with solution derivations that address this array of requirements calls for a decision making process with supporting tools that will help stakeholders select a satisfying solution for each mission area

2.1. Architectural Selection Methodology

The System-Aware cyber assessment methodology described here was designed to be an iterative process that relies on inputs from a range of stakeholder communities. In order to ensure that the information being used is as accurate and certain as possible, we found that it is imperative to ask individuals a variety of pertinent questions that were appropriate to their backgrounds and areas of expertise. This can be accomplished by initially dividing the stakeholders into three distinct groups:

Red Team - The red team is made up of individuals with knowledge of cyber-attacks and potential threat agent classes. Their work is focused on developing candidate attack vectors and assessing the effectiveness of the proposed design patterns.
Blue Team - The blue team can consist of two sub teams, one consisting of security solution designers and the other system users for the system being protected. Their responsibilities include identifying and prioritizing the critical system functions to protect, as well as determining which security design patterns can be implemented on which system functions.
Green Team - The green team, which is comprised of experts in system cost analysis and adversary capability, analyzes costs, to both the attacker and defender, for candidate architectural solutions.

The steps of the architectural selection methodology are as follows:

Step 1: Define the Variables and Relationships Within the System to be Protected
The initial step of the methodology is focused on framing the problem to ensure that all participants in the process are on the same page regarding the system to be protected. The process begins by identifying the critical functions of the system and defining the variables and influence relationships within that portion. Step one is to be performed by the blue team and is intended to outline the expected functionality of the system with minimal defensive strategies implemented. At this point, a system influence relational diagram is constructed using directed acyclic graph (DAG) notation. This

diagram is created for the system without the consideration of a cyber-attack to ensure that everyone involved in the process is in agreement on the system's structure and components before addressing the additional complications related to an adversary.

Step 2: Identify the Possible Paths an Attacker Could Take to Exploit the System
Step two introduces one of the issues that make this specific problem very complex: an intelligent adversary. While the system influence relational diagram represents a system where success may be compromised by random failures, the cyber security architecture selection problem introduces concerns where the decisions made by an active player in the system can also compromise mission success. In step two, the red team is tasked with constructing an attack tree for the system functions identified in step one. By looking at the system from the perspective of an adversary, attack trees can be utilized to understand the possible paths an attacker could take to exploit a specific feature of the system.

Step 3: Determine the Subset of Attack Actions Most Desirable to an Attacker
Considerable analysis can be conducted after the construction of an attack tree. However, rather than focusing on quantitatively calculating the probability of success for a specific attack path, as is typically done in attack tree analysis, the analysis included in this framework considers a more qualitative, abstract metric space. In step three, the green team develops a set of variables that can be used to assess the difficulty of a particular attack path. These variables are called behavioral indicators and can include, but are certainly not limited to, resources such as technical ability, time, manpower, money, equipment, facilities, presence of an insider, and access to system design information. These variables are used to make two separate types of judgments: leaf node assessments and adversary profile construction.

Step 4: Identify Appropriate Defensive Actions and Their Impacts on the Attacker
After the red and green teams have identified the actions that an adversary would need to take to successfully execute an attack and the subset of those that are most attractive to a particular adversary, the blue team can then determine which of their existing defensive actions may be appropriate. This assessment addresses both the criticality of disruption of the system function that is attacked and the complexity of the required defense mechanisms. The suggested relational methodology relies on the assumption that a portfolio of design patterns has already been developed—either by previous blue teams or by an external group no longer involved in the process. If the current blue team was not responsible for developing the set of design patterns, it is assumed that they have access to the portfolio and the have the necessary knowledge regarding the meaning of each design pattern.

The goal of step four is to select design patterns from the existing portfolio that could be implemented to make the actions captured in the leaf nodes of the attack tree less desirable to the attacker. This can mean increasing the difficulty, cost, or probability of detection to the adversary or lessoning the consequences felt by the defense in the case of a successful attack.

Step 5: Evaluate the Impacts of the Selected Potential Actions on the Defense
While step four captures the design patterns' impacts on the adversary, step five transitions to evaluating how those same choices impact on the performance of the system to be. The green team is able to apply their second class of intelligence information here: cost analysis estimates for the defensive solution choices. At this point, each of the design patterns selected in step four is evaluated in regards to implementation cost, lifecycle cost, and collateral system impacts. The green team is responsible for estimating the monetary cost of a solution, but the blue team also adds input on a solution's collateral system impact here. The blue team performs the evaluation of the solution's collateral impacts since they have knowledge regarding the system, how it will be used, and what

impacts are unacceptable. Any solutions that are deemed to be beyond the allocated budget for System-Aware security or introduce unacceptable impacts on system performance can be eliminated from further analysis at this point.

There is one deliverable for this step: a reduced list of possible defensive choices, filtered from the original existing design pattern portfolio, to only those that increase the difficulty for the considered attacker while still remaining at an acceptable impact to the defense.

Step 6: Weigh the Security Trade-offs to Determine Which Architectural Solutions Best Reverse the Asymmetry of a Potential Attack

The goal of the sixth and final step is for all three teams to participate in a collaborative discussion regarding the security trade-offs that exist with the potential choices determined in step five. While each defensive solution remaining after step five provides some potential security benefit, has an acceptable impact on the system being protected, and fits within the allocated budget, the exact mixture of security solutions to an integrated budget is determined in this step.

2.2. Model-based Approach to Architectural Selection

The Phase 2 activity in architectural selection focused on investigating the hypothesis is that a scalable and agile approach to the mission-focused architectural selection problem can be found by making use of two structured modeling approaches:

System Models using model-based systems engineering tools and languages such as SysML, which provides a mechanism for detailed description of of system structure, functions, and information flows in a searchable data format. The models allow one to capture the relationships between functional system entities and to recognize patterns (data, dependence, control) within the system. They also facility representation of the system attack surface, to mitigate the danger of under modeling system attacks. SysML models can be used represent the initial system "as-is" with minimal defense and again with possible security solutions implemented, perspective which can demonstrate the value of solutions in the context of the whole system and an understanding of the complexity added to an attack by particular defenses.

Attack Trees to identify possible paths an attacker could take to exploit the system. These models use assessments of the attack actions and the attackers' capabilities to determine the subset of most preferable actions.

2.2.1. SysML for System-Aware: Concept of Employment

In System-aware Cybersecurity, the system of interest is *the mission*. Such a mission could be as large as Joint-level Global Persistent Attack (GPA) or as limited as a specific unit's execution of Close Air Support (CAS). As such, the highest-level concept in the model is the SysML block *Mission Context*, representing the highest level of analysis at a given time, as the comparison of multiple missions is not captured. The high level components of the mission, such as platforms, operators, and sensor assets are modeled as *parts* of the *Mission Context*. When an attack or defense is introduced to the model, it is added via *specialization* of the original mission, with a number of custom modifications possible beyond the feature set provided by the SysML specification.

To model this mission-centric perspective, we chose to first decompose the mission-of-interest's system-of-systems in terms of requirements, structure, and a limited sense of behavior. Within those broad categories, we predominantly employed the concepts of requirements, use cases, blocks, ports and flows, generalization/specialization, and activities.

While a significant portion of the SysML language specification was exercised to model the mission-of-interest, many features were beyond the scope of this effort, or would be unrealistic in a representative setting. Where SysML activities were employed, they were used in a limited manner: highly detailed activity diagrams can be used to represent entire security assessment frameworks[1]. SysML constraints were not employed for our analysis, but future work could include capturing the numeric artifacts of external modeling activities, such as attack tree values and component costs, via this mechanism. Similarly, with more advanced analysis SysML state machines would be very appropriate for representing the requirements as state of the system-of-interest, from the mission to subcomponent levels.

2.2.2. SysML4SystemAware

Among the features of SysML most interesting for integration with other models is its extensibility through stereotypes. A stereotype can apply new features to nearly any part of the system model, and can reference information inside the model or external to it. For System-Aware, we have chosen to model the analysis process itself within the system model, explicitly capturing the evolution of the mission context. To achieve this, the stereotypes in Table 1 were created in the SysML4SystemAware profile:

Stereotype	*Target*	*Slots*	*Description*
CaseRoot	Block		The root structure of a system of systems evaluation: usually the mission (or a specialization)
HasSystemAwareID	Element	systemAwareId	A tool-independent identifier to allow for cross-model identification, *a la* CAPEC
ReplacesPart	Block	part	Replaces an existing part of a block within a mission context
ModifiesInformationFlow	Block	toFlow addsGeneralization	Replaces an existing flow between two blocks within a mission context

Table 1 - Stereotypes in the SysML4SytemAware profile.

2.2.3. Overview of Attack Trees

Inspired by research in the reliability area, Weiss [1] in 1991 and Amoroso [2] in 1994 proposed to adopt a tree-based concept of visual system reliability engineering to security. Today, threat trees [2,3,4], threat logic trees [1], cyber threat trees [5], fault trees for attack modeling [6], and the attack

[1] Mohamed, Otmane Ait, and Samir Ouchani. "Probabilistic Verification of Security Properties in SysML Activity Diagrams."

specification language [7] can be subsumed under attack trees, which are AND-OR tree structures used in graphical security modeling. The name attack trees is often attributed to Schneier [10].

In the attack tree formalism, an attacker's main goal (or a main security threat) is specified and depicted as the root of a tree. The goal is then disjunctively (OR) or conjunctively (AND) refined into subgoals. The refinement is repeated recursively, until the reached sub-goals represent basic actions. Basic actions correspond to atomic components, which can easily be understood and quantified. Basic actions are called leaf nodes. Disjunctive refinements represent different alternative ways of how a goal can be achieved, whereas conjunctive refinements depict different steps an attacker needs to take in order to achieve a goal.

Similarly to decision trees and fault trees, attack trees are represented by a diagram with a single root node at the top that represents the overall goal or objective of the adversary (i.e., a successful attack) [4, 8]. Attack trees are constructed from the perspective of the adversary. When building the tree, initially the modeler focus on what the attacker wants to achieve and the various ways to accomplish it, and not how to best defend the system [4].

The root node is usually a broad goal of the attacker and as such, it does not provide much information as to how an adversary might execute an attack, so the tree continues to branch down, breaking the root node into smaller steps. This process of decomposing the intermediate goals continues until the leaf nodes (which cannot be further broken down) are reached. The leaf nodes in an attack tree represent specific, concrete actions that an adversary could take and that could lead to a successful attack if executed in combination with others in the tree. Figure 3 shows a very simple attack tree, with the purpose of demonstrating the notation and illustrating the different types of nodes.

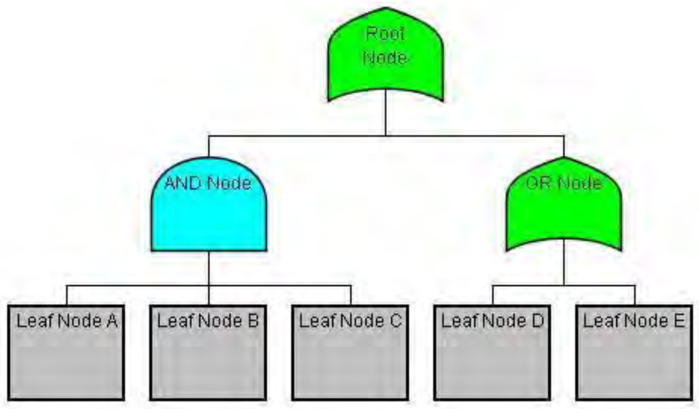


Figure 3 - Simple Example Attack Tree to Demonstrate Notation

Since the root node in Figure 3 is an OR node, the attacker only needs to execute one of the input paths following level: executing the conjunctive actions on the AND node path or executing the disjunctive actions on the OR node path - either path will be sufficient to accomplish the overall goal. If the attacker chooses to attempt the AND node path, they must complete all three of its associated leaf nodes (A, B and C) - no order is required in a simple AND. On the other hand, if they choose to take the right hand path of the tree, they only need to complete one action (either leaf node D or E), since they are joined by an OR node. This means that the attacker has three different paths to traverse the tree (known as attack scenarios): (1) nodes A, B and C, (2) node D, or (3) node E. The successful completion of any one of those sets will allows the adversary to reach the root node at the top of the tree and accomplish their overall goal.

After the tree has been constructed and the set of leaf nodes has been identified, analysis can be carried out to identify a subset of the most likely attack scenarios. This includes making assessments for several different adversary profiles regarding their preferences and the capabilities they are expected to possess and assessing each attack action in regards to those same behavioral indicator variables. This information can then be used to prune the attack tree for a specific attacker to identify the subset of their most preferred attack actions.

Quantification of attack scenarios with the help of attack trees is a very active topic of research, and the attack tree tool we are using (SecureITree)[2] allows for various forms of quantification. A simple procedure for quantification of attack trees is based on a bottom-up algorithm. In this algorithm, values (enumeration of attacker attributes) are provided for all leaf nodes and the tree is traversed from the leaves towards the root in order to compute values of the refined nodes. Depending on the type of refinement, different functional operators are used to combine the values of the children. This procedure allows one to analyze simple aspects, such as the costs of an attack, ease of attack, propensity of attack, the necessary skill levels, resources, etc… Whenever more complicated attributes, such as probability of occurrence, probability of success, risk or similarity measures are analyzed, additional assumptions, for example mutual independence of all leaf nodes, are necessary, or methods different from the bottom-up procedure have to be used. The tool we have chosen for this exploratory effort is very flexible with regard to different quantification strategies and threat agent profiling.

By incorporating factors/attributes that characterize the adversary into the attack tree analysis, judgments can be made about which attack actions may be more desirable to the adversary [11]. Attack scenarios, which are near or beyond the attacker's perceived capabilities/resources, are less preferred than attacks that are perceived as simple and inexpensive. Additionally, the extent to which an attack satisfies the adversary's objectives also affects their choices: actions that are both within the adversary's capabilities, and which satisfy their goals (cost, ROI, etc…) are more likely to be perused than those that do not. This notion reflects the *propensity* of the attacker.

In almost all attack tree model formulations, the direct interaction between the adversaries and the defender's system occurs at the lower levels of the attack tree - the leaf nodes. Therefore, it is useful to associate metrics with each leaf node operation describing the resources required of the adversary. The types of resources that characterize the adversary or influence them are for instance, the cost of the

[2] http://www.amenaza.com/SS-what_is.php

attack, technical ability, specific knowledge, time needed, and noticeability of an exploit are all potential influence factors. Almost always, values or information for these parameters are obtained from subject matter experts (SMEs) who provide estimates based on their expert understanding of the activities of the threat agent and the system to be defended.

2.2.3.1. Prerequisites of an Attack

Three conditions must be present in order for an attacker (also known as a threat agent) to carry-out an attack against a defender's system.

1. The defender must have *vulnerabilities or weaknesses* in their system.
2. The threat agent must have sufficient resources available to exploit the defender's vulnerabilities. This is known as *capability*.
3. The threat agent must believe they will benefit by performing the attack. The expectation of benefit drives motivation.

Condition 1 is completely dependent on the defender.

Whether condition 2 is satisfied depends on both the defender and the threat agent. The defender has some influence over which vulnerabilities exist and what level of resources will be required to exploit them. Different threat agents have different capabilities.

Condition 3 mostly involves the attacker. It represents the motivation to carry out the attack. The defender may have a role if their actions provoke a threat agent to carry out an attack.

In short, the threat agent and the defender jointly contribute to the conditions that determine whether an attack can occur or is likely to occur. Proper attack analysis requires that we examine all three conditions above in order to predict the behavior of adversaries and the likelihood that an attack will occur. Understanding these factors also provides insight into effective ways of preventing attacks.

2.2.3.2. Modeling Threat Actors

A Threat Agent is a group of people, or outside individual, or insider likely to cause harm to a system. e.g., hackers, industrial spies, disgruntled employees, etc. [3, 12]. Various combinations of these individuals reflect a *specific profile threat*. Threat agents perform actions based on motivations, such as monetary gain, need for critical timely information, strategic advantages in the marketplace, etc… Thus we make the reasonable assumption, that attacker motivations are related to attacker benefits, that is, adversaries make decisions on the basis of *cost-benefit-risk*. For a given threat agent, the *cost* of obtaining vital information could be in the form of capital and human investment, *benefit* could be advantage in the marketplace or the acquisition of vital defense information, *risk* could be noticeability or attribution of the attack.

Threat Agents are constrained by their capabilities and the resources available to them. If these limitations are known, we can characterize them in a way that allows us to approximate the threat. Since all of the direct interaction between the adversaries and the defender's system occurs at the leaf nodes, it is useful to associate attributes with each leaf node operation describing the resources required of the adversary. The types of resources examined should be factors that influence the behavior of the adversary according to the *cost-benefit-risk* tenant.

In most Attack Tree tools and methods (including the tool we use), the composition of these threat agent attributes is realized by *threat agent utility functions [4, 12]*. A utility function can be as simple as likert scale(high to low attribution of threat agent trait) or something more complex such as curve that indicates, for example, attacker motivation as function of monetary benefit to the attacker. Due to variances in human behavior within a threat agent class, no curve or function will ever be a perfectly accurate description of a specific threat agent's decision-making process, however, utility functions allow the analyst to approximate threat agent behaviors against the system in ways that allow "what if" analysis, and "threat agent specific" analysis.

Every attack requires the threat agent or adversary to expend a variety of resources. The analyst chooses specific types of resources to include in the threat agent model based on the degree to which they characterize and influence the adversary's ability to perform the various attack scenarios within the attack tree. These resources can include money, raw materials, technical ability, time, knowledge of the system, and a willingness to be noticed. Even though everyone might be forced to spend the same amount of a resource to perform a specific attack that does not mean that they are equally willing or able to do so. The availability of resources varies from threat agent to threat agent. For instance, a relatively poor teenage computer hacker might consider $100 to be of considerable value and be strongly disinclined to part with it without a substantial benefit. On the other hand, a well-organized cyber-criminal threat agent might regard $100 as pocket change. However, the time-crunched cyber-criminal adversary would be far less willing to part 400 hours of his or her precious time than the bored adolescent who is happy to pass the hours away trying to crack a computer system. So, the *human value* of resources is important with respect to characterizing the agent behaviors.

An example of utility functions would be describing the spending propensity of juvenile hacker threat agent. For example, suppose we created a profile of the juvenile hacker that specified a financial limit of $50. This simplistic profile asserts that the juvenile delinquent is completely and equally willing to spend any sum between$0 and $50, but that they would be unwilling to spend $51. This seems unrealistic in terms of human behavior, as thresholds for "willingness to something or not " are more gradual, than hard. Figure 4 below shows the simplistic profile on the left (which is used in most scoring approaches to security today). One the right is a more representative function of a juvenile hacker's propensity to part with his money.

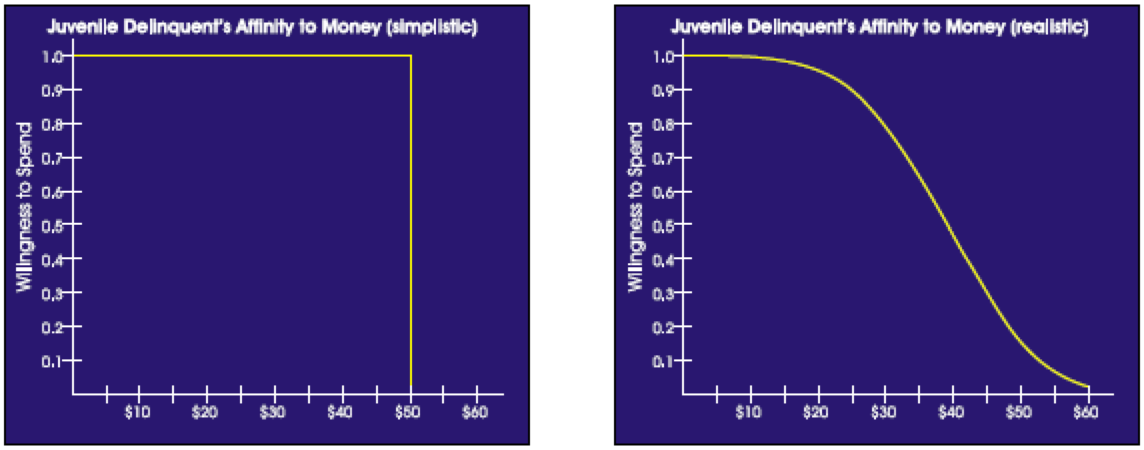


Figure 4 - Utility Functions for Agent Profiles

2.2.3.3. Modeling Victim Impacts

In addition to characterizing Threat Agents. It's also possible with to characterize victim impact with Attack Trees [1,3]. From the victim's perspective, there is also a concern about the consequences of an attack, and how much it will damage or cripple the mission– the perceived impact to the victim. The victim can be organization, a group, a person, or a service entity in a nation state. Attack trees model victim impacts by calculating the amounts and types of losses that a victim will incur as an attacker achieves particular states (nodes) in the attack tree. In most cases, the victim impacts are cumulative as the attacker traverses an attack path through the tree. The overall impact of an attack scenario is calculated by summing the damage associated with the leaf nodes specified in the scenario, and then moving upward through the intermediate nodes traversed by the attack scenarios steps until the root node is reached (full success).

The attack tree tool we used for this effort (Amenaza Secure*IT*ree) has a well formed "actor definition utility" model that allows utility, affinity, and Likert functions to be used to describe threat actor and victim attributes [3].

2.2.3.4. Issues with Static Attack Tree Methods

Like most risk analysis methods, the reliance on Subject Matter Experts (SME) to convey opinions on threat agent capabilities/resources, and to simultaneously relate this knowledge with respect to the target system can impart uncertainty in the analysis process. Understanding or assessing vulnerability of target system requires detailed information from many perspectives. While using SME's to help identify potential vulnerabilities is helpful, it is usually not sufficient. For example, a SME may know a good deal about threat actor capabilities, but they may not understand the target system well, which could lead to over or under estimating the defenses required to augment security.

The final attack tree is often produced only through a process of review and consensus building between the SME and security engineers. Even after a consensus is reached, it is unlikely that the analysis results will be complete, consistent due in part to the informal processes used as the basis of the analysis. In fact, the lack of precise models of the system architecture at various levels of abstraction, often forces the security analysts to devote much of their effort to gathering knowledge about the system architecture and system behavior and embedding this information in document based artifacts.

In fact our previous efforts conducting cyber-vulnerability assessment within the System Aware Architectural Selection Framework was a manual consensus building process between Subject Matter Experts which took many person-hours of time with significant amount of uncertainty in the characterization of some of the identified vulnerabilities and exploits. It was by this process, that we realized that a more promising way forward is through system level modeling.

Another issue with most attack tree formalisms (but not all) is they are largely static. Meaning they can not take into account dynamic behaviors such as; (1) threat actor actions based on state of the system, (2) phased attacks over time, or (3) sequence dependencies with respect to steps in a attack. Dynamic fault tree tools have solved many these issues, such as Galileo, but there are few mature dynamic attack tree tools in the literature. Dynamic modeling of system behaviors can be effectively handled in a functional system level-modeling environment, such as UML or SysML or AADL.

Therefore for this effort, we established a hypothesis that the issues discussed above could be significantly improved by basing cyber vulnerability analysis and mitigation on *system level models*. We call this approach *Model Based Cyber Vulnerability Assessment*, which is a derivative of Model Based Systems Engineering paradigm [13]. Part of this effort focused on what aspects of model based engineering can inform attack-tree security models to improve consistency of models, realism of attack patterns and discovery of vulnerabilities. The other aspect of this effort was to develop a basis for integrating attack tree disciplines and system modeling processes in an effort to explore a more holistic view of the system - to capture greater insight into vulnerabilities, unforeseen relationships between system functions, navigating the multi-criteria decision space, and how to craft defenses that are cost effective.

To facilitate model based cyber vulnerability assessment, we chose the modeling language SysML to provide the framework for capturing system aspects such structure, platforms, component behaviors, mission context. Models realized in this framework allow subtle, but important relations and connections between mission, architecture, functions, and users (misusers) to be explored in systematic way.

2.2.4. Overview of the Systems Modeling Language

As we noted above, enhancement of the architecture selection approach from previous System-Aware activities is the move toward a *system model* for the system of interest. While not necessarily a simulation of the system of interest, it should capture the key terms, interrelationships, data types, requirements, and other knowledge of the system that may be outside a single engineer's purview.

The Object Management Group (OMG) Systems Modeling Language (SysML) provides a data model and a visual syntax for defining a *system model*, the "primary artifact of model-based systems engineering"[3]. In a generic sense, a SysML model sits at the middle of a complex engineering effort, providing a central point of integration for the perspectives of many domains of engineering. Such a model describes the "requirements, structure, behavior, allocations, and constraints"[4] of a system of any level of complexity. This facility to work at multiple scales of complexity comes from SysML's heritage as a descendent of the OMG's Unified Modeling Language (UML), the primary means of modeling object-oriented (OO) software.

SysML's key departure from UML is in the core vocabulary, the *block*, vs UML's *class*. A block represents a nameable part of a system, and may have *properties*, which may be other *blocks*, *constraints*, atomic *values*, etc.

2.3. Integration Principles

To bring together the system- and attack tree-modeling disciplines, a number of technology selections were required. In each case, a balance needed to be struck between performance, ease of use, openness, supportability, isolability, quality of collaboration, and cost.

2.3.1. Authoring tools

Model authoring tools were chosen primarily for their productivity and support of underlying model concepts. While an initial emphasis was put on creating an open source tool chain, this proved too difficult to rectify with productivity. Because of this, the next most important feature was openness of

[3] Friedenthal, Sanford, and Alan Moore. *A Practical Guide to SysML the Systems Modeling Language*. [2nd ed. Amsterdam: Morgan Kaufmann, 2011. Print. pp. 528

[4] "OMG Systems Modeling Language (OMG SysML) Version 1.3." *Object Management Group*. 1 Apr. 2012. Web. 5 Jan. 2015. <http://www.omg.org/spec/SysML/1.3/>.

data import/export, such that another tool could be adopted at a later date without sacrificing historical data.

2.3.2. Attack Tree Authoring

While many tools exist for attack tree modeling in the academic world, only a few exist in sufficient maturity and stability in the commercial world, notably Amenaza Secure*I*Tree and Isograph Atacktree+. In this effort we choose to utilize commercial tools as they are mature and supportable. The Attack Tree authoring tool we used for this effort was Amenaza Secure*I*Tree. Secure*I*Tree is a widely used commercial, mature risk based attack tree tool. Among the features that we liked in Secure*I*Tree were the underlying models for quantitative analysis. The underlying quantitative model is roughly based on four aspects:

- The ability to characterize the threat agent profile in several ways that are complementary to open source intelligence data or information,
- The ability to instantiate leaf nodes with system level exploit details (low level) and at the same time instantiate mission level data (the effect of the attack) at higher level nodes in the tree
- Risk based models, the ability to capture impact of an attack on a system
- Employs mature and stable fault tree solution methods to prune the attack tree and calculate metrics such as *ease of attack, propensity of attack, desirability of attack, etc...*
- The ability to write java plug-ins, export the Attack Tree model in ATML (Attack Tree Modeling Language), and export of data to other tools and models.

We found these features to be useful in our preliminary development efforts, and for our future development efforts as well.

2.3.3. SysML Authoring

Initially, we had a strong desire to make use of the current best-of-breed free/libre open source authoring tools for SysML. The strongest FOSS contender, Polarsys *Papyrus*[5], was initially attempted and found to be too cumbersome for the types and rapidity of modeling desired for the System Aware work. The other FOSS tool inspected, Modelio *SysML Architect*[6] did not contain some of the required features, namely SysML requirement. With the FOSS tools exhausted, the team was left with those tools most widely employed in DoD-related work by the team in the past, namely NoMagic *MagicDraw with SysML Plugin*[7].

2.3.4. Data Types

At the heart of the model-driven System-Aware approach is the model data itself. Generally, a format should be stable, standards-based and accessible in different analysis environments.

2.3.5. Attack Tree Modeling Language

Due to the relative size and niche nature of the attack tree modeling community, no data standard exists for interchange of attack tree data between tools. The nearest we could find was the *Attack Tree Markup Language*, an XML-based export format of Secur*I*Tree. This format is sufficient for basic transmission of knowledge, but a richer format would be required for serious integration into a

[5] Papyrus. Polarsys. Web. http://polarsys.org/solutions/papyrus

[6] Modelio SysML Architect. Modeliosoft. Web. http://www.modeliosoft.com/en/modelio-store.html?sobi2Task=sobi2Details&catid=12&sobi2Id=37

[7] SysML Plugin. No Magic. Web. http://www.nomagic.com/products/magicdraw-addons/sysml-plugin.html

knowledge management environment. Linking the Attack tree and SysML model world is not straightforward – very different SW structures. However, Amenaza is working an external database update solution for future versions of the Secure*IT*ree Attack Tree tool. Basically, the external database solution would enforce consistency between SysML models and Attack tree models whenever a node in the Attack tree is changed (created, modified, deleted) in SecurITree, you will be able to have an entry written to a logfile for parsing at your convenience (like updating a model data base, for instance). You will also be able to ask SecurITree to invoke a user defined Java function that will perform these database operations. These additional features would facilitate tighter integration between SysML models and Attack Tree models; however, at present we do not know when these features will be released. Ultimately, extracting records from the database will enable support for the workbench decision-maker tools.

2.3.6. SysML XMI

All SysML tools use a dialect of the XML Metadata Interchange (XMI)[8] format. As of SysML 1.3, a canonical XMI specification exists for interchange of model data between SysML tools[9]. However, most authoring tools bundle their model and visual descriptions together, and a canonical diagram XMI format will not be available until broader adoption of SysML 1.4[10]. The means the working system modeler cannot yet use the canonical format as their format of record. We were left with working with the specific XMI dialect of MagicDraw, but were careful to craft any parsing mechanism in a way that would lend itself to later adoption of the canonical spec.

2.4. Data and Model Consistency

To make use of data from different models, a semantic graph was selected as the integration model. This allows common features, such as the names of things, to be represented in a common format, while the specifics of each model could be robustly captured. The specific method adopted, named graphs as defined by the Resource Description Framework[11], offers rich reuse of concepts, where each assertion can be described by a statement of the form, *"According to <source>, the concept <subject> has been known to <predicate> the concept <object>."*

2.5. Visualization

Beyond the canonical model authoring views of attack trees and SysML provided by the chosen tools, the two models are of sufficient complexity to be treated as data, and have data visualization techniques brought to bear against them. In this case, we have chosen Data Driven Documents (d3)[12], which provides an open source, standards-based data visualization suite. With the techniques available in d3, such as interactivity, animation and object constancy[13], different facets of the union of the models can be represented in a form less tied to a specific datum's model of origin, to provide a more approachable experience for the non-modeler or the modeler working outside their domain of expertise.

[8] "XML Metadata Interchange" Object Managment Group. Web. http://www.omg.org/spec/XMI/

[9] "Model Interchange." Model Interchange Wiki. *Object Management Group*. Web. http://www.omgwiki.org/model-interchange.

[10] "SysML 1.4 Beta". *Object Management Group*. Web. http://www.omg.org/spec/SysML/1.4/Beta

[11] "RDF 1.1 Concepts and Abstract Syntax." *W3C*. Web. http://www.w3.org/TR/rdf11-concepts

[12] "D3: Data-Driven Documents" Michael Bostock, Vadim Ogievetsky, Jeffrey Heer IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011

[13] "Object Constancy". *Mike Bostock*. Web. http://bost.ocks.org/mike/constancy

2.5.1. Configuration Management

To enable the fluid interchange of data between multiple modelers at disparate sites, we used a system such as could be envisioned in a multi-level security environment: systems models would likely exist at a lower classification than the possible exploits discovered by red teams. Using an offline-capable distributed version control system, git[14], our system modeler and attack tree modeler were able to confidently pass data from low to high levels.

2.5.2. Integration Environment

With the model data under configuration management, it was important to be able to load all analysis and data in a consistent, repeatable environment. For this purpose, Vagrant[15] was chosen, which can create an executable environment from a simple text description. Inside of this virtual environment, CentOS Linux[16] was selected, as either it or its equivalent, RedHat Enterprise Linux, would be likely operating systems for deployment of integrated workflow system. For the actual analysis environment, we selected the IPython Notebook[17], which combines the breadth and depth of the Python programming language, along with a rich, browser-based interactive execution and documentation environment. Because of the focus on ease of use, we call this total integration environment the *dashboard*, even though it may be comprised of multiple Notebooks.

The current integration approach is limited in one sense by this architecture: the current knowledge graph can only scale to the scope of the memory (RAM) of the environment. A more robust system making use of a graph database would increase the potential size of the analyzed environment exponentially.

3 Architectural Assessment Workbench Concept

The execution of the architecture assessment process can be captured by a discrete number of activities executed within a tool. These steps may have data dependencies between them, but for the most part can be executed in any order, and in fact are improved through successive iteration. Figure 5 below is a conceptual view of the workbench concept (research still ongoing). The composition and functionality of the workbench is designed to support the System-Aware Architectural selection methodology. We discuss the ongoing capabilities of the workbench below.

[14] "Git". Git community. Web. http://git-scm.com/

[15] "Vagrant". Hashicorp. Web. https://www.vagrantup.com

[16] "CentOS". CentOS Project. Web. http://www.centos.org

[17] Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007

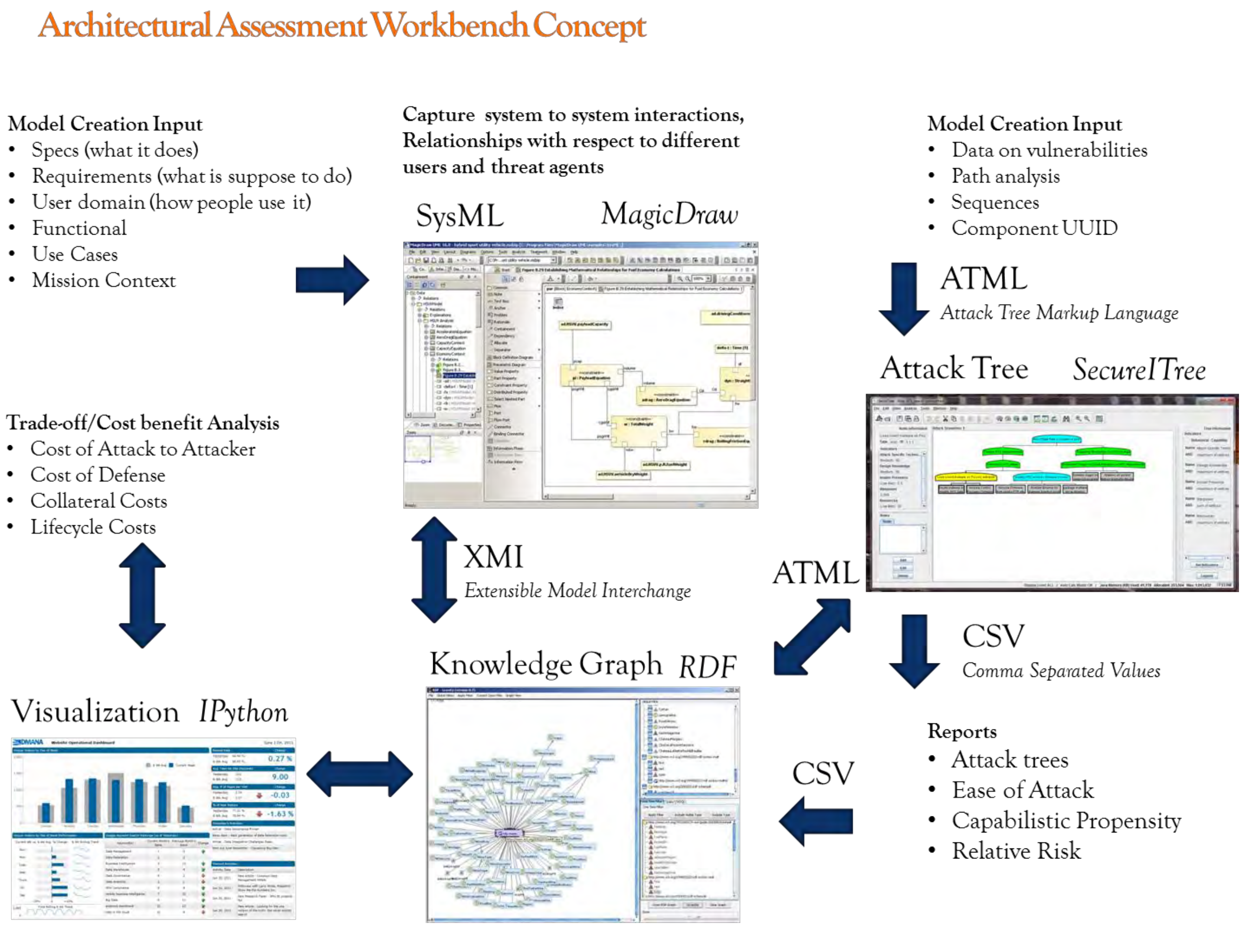


Figure 5 - Architectural Assessment Workbench Concept

3.1. Initiate System Model

Using the existing data, such as OEM documentation, military doctrine and subject matter expertise from operators, commanders and other decision makers, we construct a SysML model of the mission context. Canonically, first the uses cases of the model are defined, followed by requirements, behavior and finally structure and connectivity. We found that parallel, iterative definition of these aspects of the system, with feedback between different users and domains, rapidly provided a sufficiently rich definition of the system.

3.2. Visualize System Model

With even the barest vocabulary of the system-of-interest in place, it is already possible to perform visual assessment of the model. We parse and load the SysML XMI into the knowledge graph, and interactively view different aspects of the data in the dashboard. As the information available improves, the graph of the model of interest can be filtered to only include those mission contexts that include specific attacks and defenses.

3.3. Identify Exploits

With access to knowledge of the flows of information through the system, the attack tree modeler can identify system functionality an adversary would like to affect, and theorize about potential vectors of exploitation.

3.4. Model Exploits

Even prior to a full description of a vulnerability, e.g. attack tree, a specialization of the base mission context can be constructed which identifies the part of the system that would be changed, and the impact that would create on information flows, such as introducing misinformation. Loaded into the dashboard, this can show additional potential impacts of an exploit.

3.5. Model Defenses

With the assistance of additional subject matter experts, or the theoretical application of design patterns, the system modeler can construct specializations of the Mission Context where a part has been created which includes one or more System-Aware techniques, e.g. sentinels. This may also introduce modifications to the requirements of the system of interest, such as the naive "*information x shall always be true and accurate*" to the more pragmatic "*information x shall either be true and accurate or be known to be misinformation*".

3.6. Compare Exploits and Defenses

Not yet integrated into the dashboard, the cost and mission impact analyses from the existing Excel assessment tool could be reproduced as near-real-time interactive visualizations, driven directly by the data products of the system and attack tree models. Further, through the greater robustness of the knowledge graph data model, findings from a specific analysis session could be more easily stored for later review.

3.7. Present to Stakeholders

Because of the flexibility of the web-centric dashboard, its content can be recombined into several open standard formats, including paged, narrative documents, standalone interactive websites, and slide-based presentations. This briefing-forward content approach keeps data fresh, and reduces copy-paste and omissions, extra rework and other sources of information decay.

4 Overview of the Baseline System: UAV autopilot and Camera System

The general architecture of the autopilot in shown Figure 6. From this figure we can see a natural grouping of relationships for the autopilot into four system types. The four major systems are:

- The Controller (red circle): The onboard processor executes all of the control laws, flight director functions, management of INS, GPS, actuators, and the communication links. The controller as represented by the red circle in Figure 6. The flight controller requires inputs from the sensory sub-system state estimator (e.g. INS, GPS, Altitude, speed) to regulate the air vehicle to a desired state, speed and position attitude. The controller also takes input from the flight director, which contains the desired trajectory reference states for the air vehicle. The flight controller uses the stored flight director information as tracking inputs, thus the flight controller is progressively issuing actuation commands to the control surfaces to minimize the error between track references and current air vehicle state and position. As such, the autopilot continuously flies the vehicle to each geographical waypoint in succession. Attacks directed to the hardware and software of the flight controller can affect the behavior of the flight controller so that it does not perform its function as intended.

- Sensory and measurement Subsystem (blue circle): The Sensory subsystem (shown as the blue circle in Figure 6) provides all of the sensed vehicle state information needed by the controller to maintain stable flight. The functions in this system include INS which provides vehicle 3-axis accelerations, angles, and velocities; GPS provides geo-reference position and velocities; Magnetometer is used to sense heading direction. Thus the total vehicle state is (ϕ, θ, ψ v_e, v_n, $v_{d,}$ a_x, $a_{y,}$ $a_{z,}$ and heading$_)$). The total sensor readings combined with the GPS information are sensed by the controller on regular time intervals (every 100ms). These classes of system attacks target the sub-systems or systems that provide data or information to the controller. In this case, the controller behaves as programmed, but some or all inputs to the controller system are corrupted. Some examples of this type of attacks include false data injection attacks to manipulate sensory data, vehicle/system component state data manipulation, and navigational waypoint data manipulation.

- The Communication system (green circle): The communication system is responsible for (1) transmitting commands to the UAV to alter flight path and (2) to receive telemetry information about the UAV in flight (green circle in Figure 6). The vehicle command signals are transmitted by the operator via a line of sight communication transceiver. The ground station communication link operating frequency is usually in one of the several designated bands (910MHZ or 2.4 MHZ are common), various signal modulation methods are used to encode the link channels. Various channels are allocated for each command or telemetry class, that is pitch, roll, yaw, and throttle will be on a separate channel than say GPS. After the onboard receiver decodes the signals from the ground station transmitter, the signals are converted to digital commands, processed by the onboard main processor. Attacks that target the communication system could affect both the vehicle and the command/control station. Telemetry data can be spoofed from the UAV, command information can be intercepted an altered, disabling of the communication link.

- Gimbal Pointing Camera system (purple circle): UAVs are predominantly used as Intelligence, Surveillance, and Reconnaissance (ISR) platforms carrying sensor payloads such as EO/IR cameras, synthetic aperture radar, signals intelligence systems, and others. Referring to Figure 6, the purple circle encapsulates the onboard Gimbal mounted camera of the UAV; capable of target tracking, scene steering and electronic image stabilization. The Gimbal system features onboard processor to control camera gimbals and stabilization effectors, a video processing system, and a communication link to send images to ground station and to the viewpoint operator station. The image operator station is capable of integrated moving map, real-time mosaicing, Path-Track, and video recording functions.

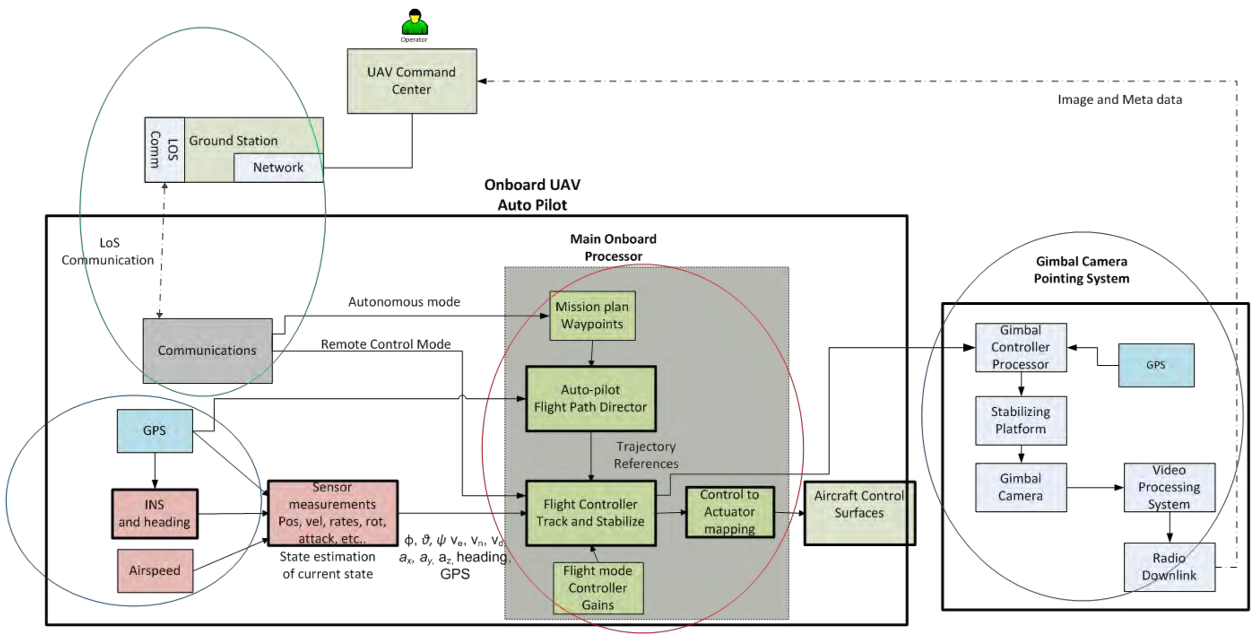


Figure 6 - General Architecture of the Autopilot

5 Case Study: A Baseline System Model for a Notional Intelligence, Surveillance and Reconnaissance Mission

The models in this section were created from a specification of the autopilot system described above. The level of effort to create the SysML models from the specification was accomplished over a 1 to 2 month period by a SysML modeling expert.

As a demonstration of the System-Aware model-based approach, we have selected a typical, yet simplified Intelligence, Surveillance and Reconnaissance (ISR) mission for the UAV system. This model was informed by previous System-Aware work, specification of the system, and subject matter expertise. Without fully qualifying all requirements of the system-of-interest, let us focus on a single, high-level requirement of the system that the platform carrying the sensor fly to the correct place, reflected in Figure 7.

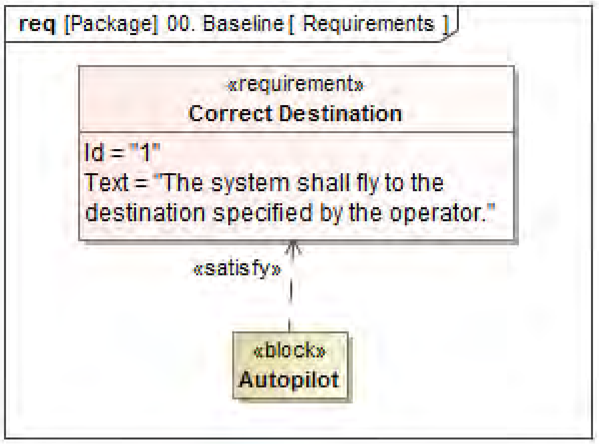

req [Package] 00. Baseline [Requirements]
«requirement»
Correct Destination
Id = "1"
Text = "The system shall fly to the destination specified by the operator."
«satisfy»
«block»
Autopilot

Figure 7 - SysML Requirements Diagram of ISR Mission

Throughout these diagrams, presented in a top-down fashion, we will show the use of many SysML concepts, as well as a few extensions provided by the SysML4SystemAware profile, as described in Figure 8.

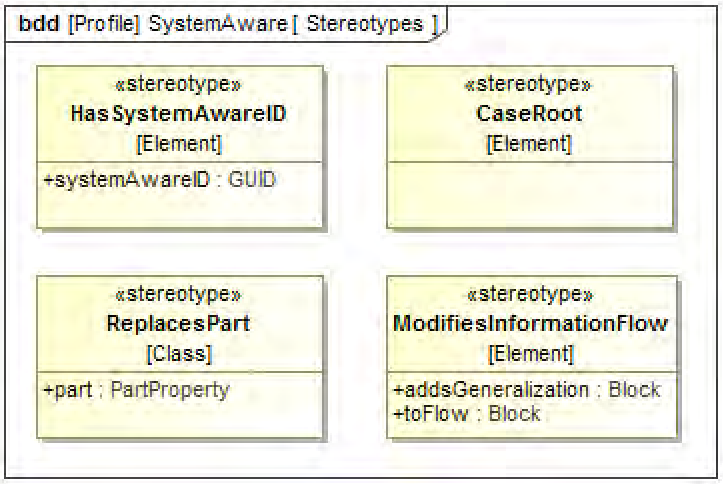


Figure 8 - SysML Profile Diagram of SysML4SystemAware Profile

At the highest level, or Case Root, the mission consists of several high level components, as shown in Figure 9. From this diagram, we can immediately see the structural components of the system, setting the stage for more detailed modeling. Future modifications of the overall context evaluation will subclass this block, and have access to all its parts.

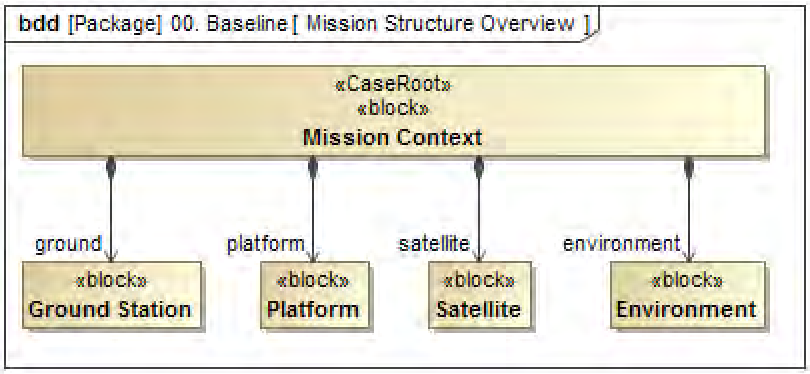


Figure 9 - SysML Block Definition Diagram of ISR Mission

With this initial understanding of the system, it is reasonable to being describing the connectivity of the components, as shown in Figure 10. From this diagram, we can immediately see the interesting relationships between high-level components without dwelling on the specifics of conveyed information.

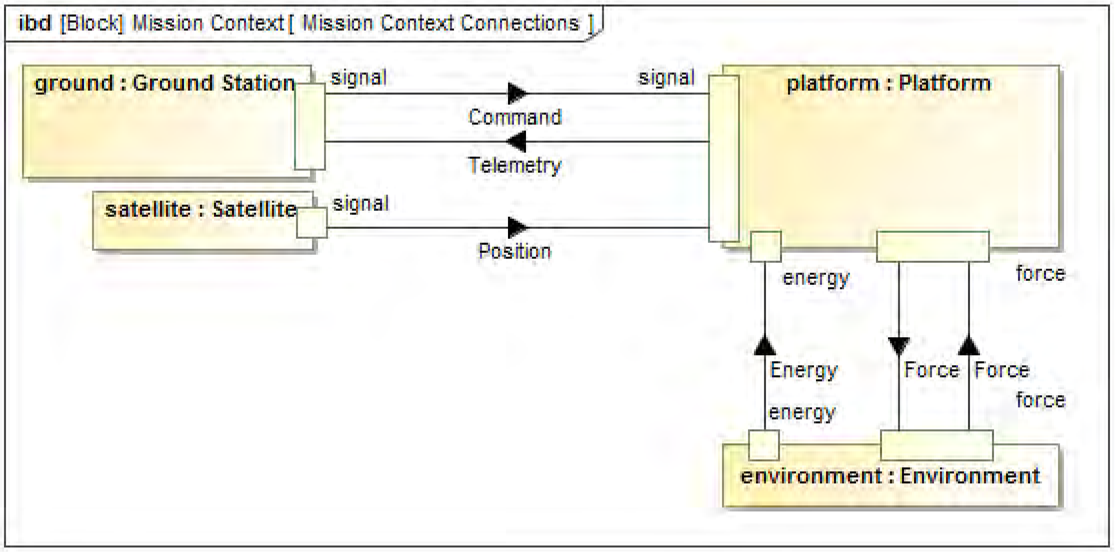


Figure 10 - SysML Internal Block Diagram of the ISR Mission

These high level components in turn are allocated the critical activities of the overall mission, as shown in Figure 11.

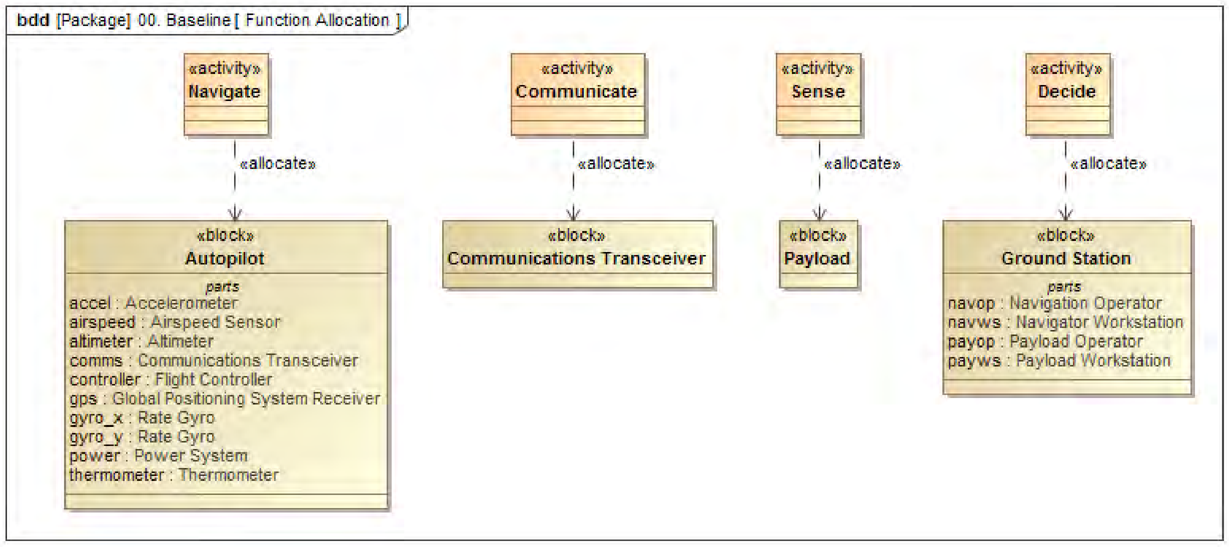

bdd [Package] 00. Baseline [Function Allocation]
«activity» Navigate
«activity» Communicate
«activity» Sense
«activity» Decide
«allocate»
«allocate»
«allocate»
«allocate»
«block» Autopilot
parts
accel : Accelerometer
airspeed : Airspeed Sensor
altimeter : Altimeter
comms : Communications Transceiver
controller : Flight Controller
gps : Global Positioning System Receiver
gyro_x : Rate Gyro
gyro_y : Rate Gyro
power : Power System
thermometer : Thermometer
«block» Communications Transceiver
«block» Payload
«block» Ground Station
parts
navop : Navigation Operator
navws : Navigator Workstation
payop : Payload Operator
payws : Payload Workstation

Figure 11 - SysML Package Diagram with Allocation of Mission Activities

Additionally of immediate interest is the flow of information between mission components, which will become increasingly relevant throughout the modeling process. Figure 12 shows the taxonomy of information types which will be used to annotate *Information Flows* throughout the system model.

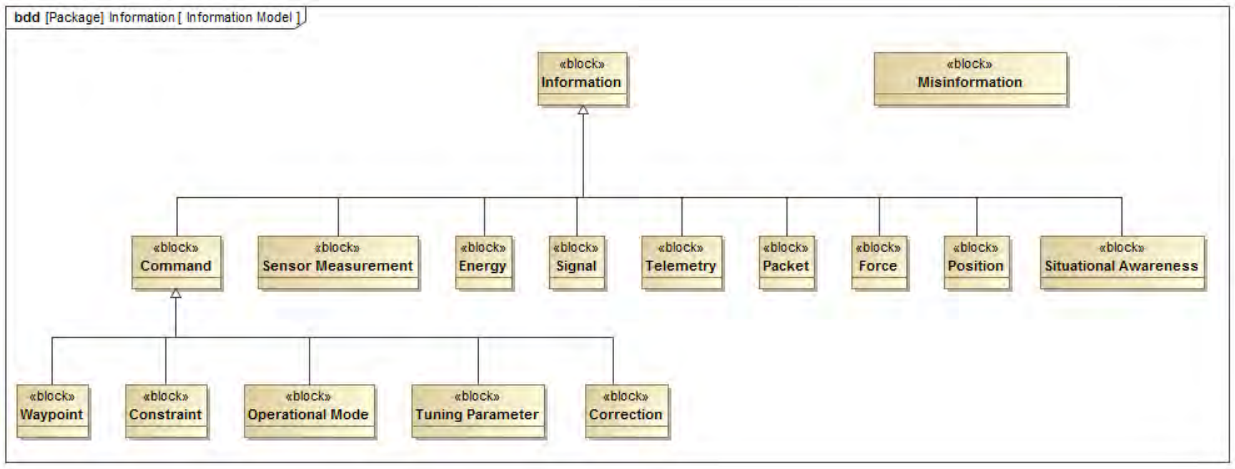


Figure 12 - SysML Block Definition Diagram of Information Hierarchy

Informed by prior System-Aware work, the platform, an unmanned aerial vehicle (UAV) is modeled at a greater level of detail. Figure 13 shows the subcomponent structure of the platform.

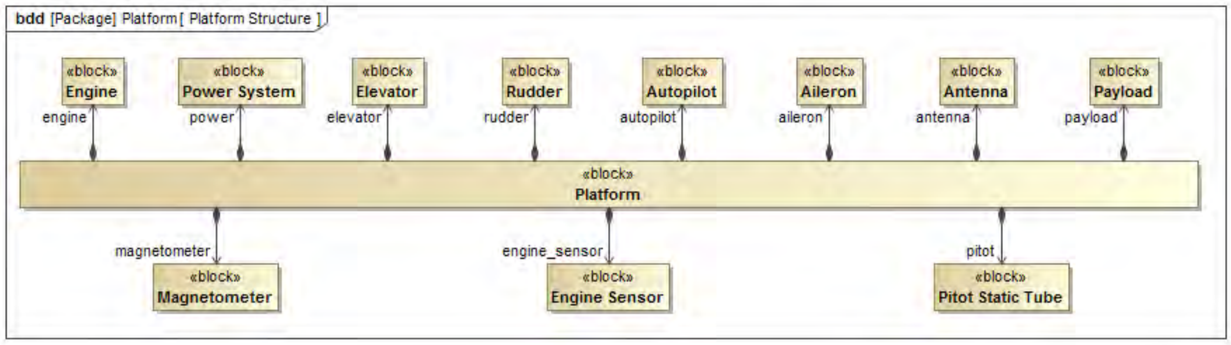


Figure 13 - SysML Block Definition of Platform Component

While the structure of the UAV is relatively simple, the interconnectedness of this subcomponent begins to show the complexity of even an isolated system. Figure 14 shows the ports and information flows, both internally and exposed at the system boundaries, that defines the UAV.

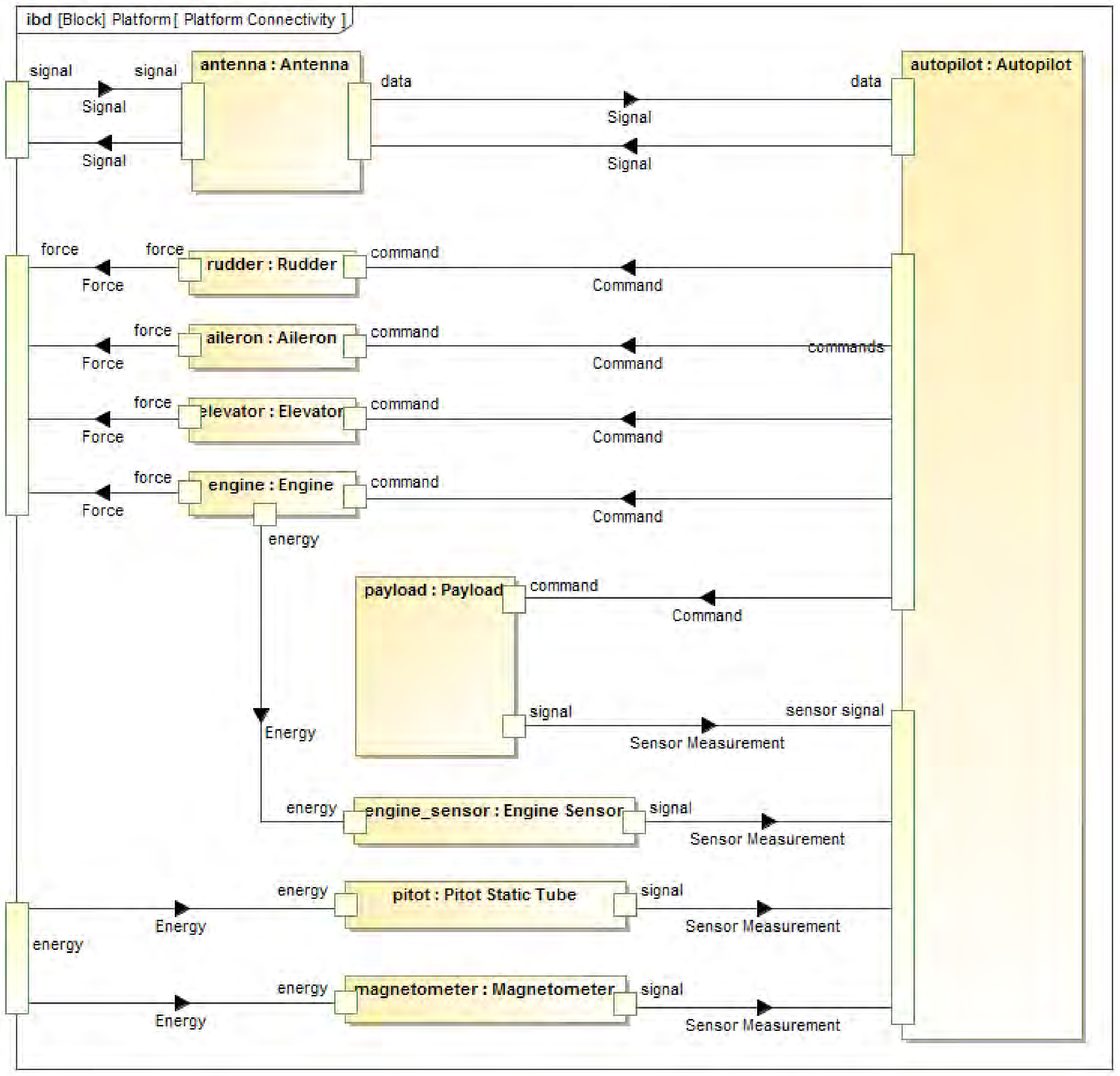


Figure 14 - SysML Internal Block Diagram of Platform Subcomponent Connectivity

At the deepest level of scale within this particular mission, the autopilot which provides the UAV's namesake autonomous operation is detailed. In Figure 15, even the structure itself is becoming complicated, as multiple levels of the subcomponent must be understood in association with one another.

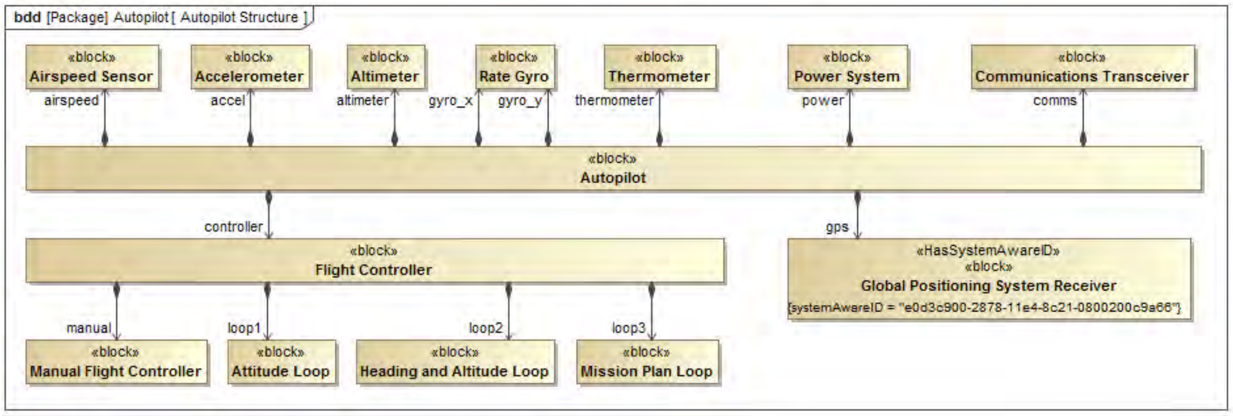

bdd [Package] Autopilot [Autopilot Structure]
«block» Airspeed Sensor
airspeed
«block» Accelerometer
accel
«block» Altimeter
altimeter
gyro_x
«block» Rate Gyro
gyro_y
«block» Thermometer
thermometer
«block» Power System
power
«block» Communications Transceiver
comms
«block» Autopilot
controller
gps
«block» Flight Controller
«HasSystemAwareID» «block» Global Positioning System Receiver
{systemAwareID = "e0d3c900-2878-11e4-8c21-0800200c9a66"}
manual
loop1
loop2
loop3
«block» Manual Flight Controller
«block» Attitude Loop
«block» Heading and Altitude Loop
«block» Mission Plan Loop

Figure 15 - SysML Block Definition Diagram of Platform's Autopilot Structure

Conflating several levels together, in Figure 16 we see the flow of information between sensors and actuators, as well as the functioning of the actual algorithm design within the flight controller.

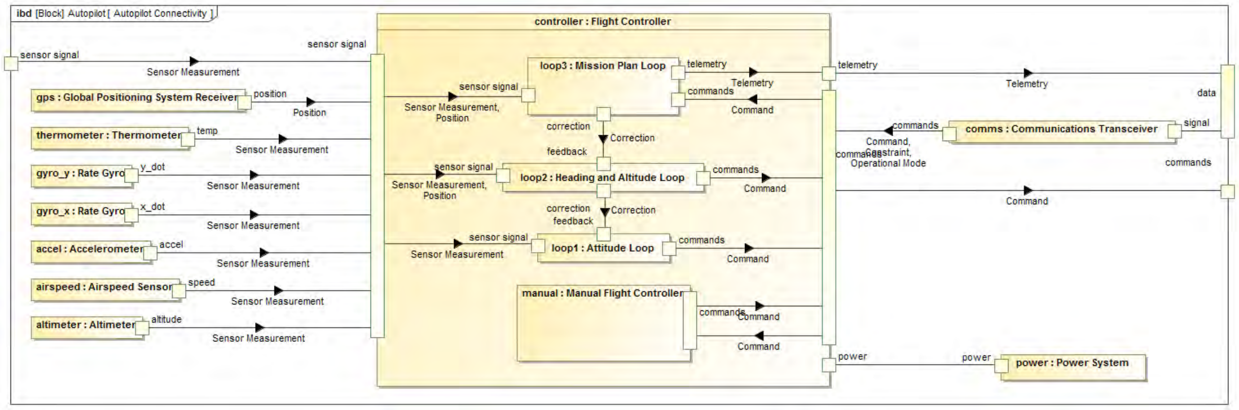


Figure 16 - SysML Internal Block Diagram of Autopilot

6 Case Study: Modeling the Effects of a GPS Attack

The System-Aware approach posits that systems will be attacked, and that the properties of the emergent system-under-attack must be understood. To reflect this, we make use of a mix of both the SysML specification for describing specialization of a system-of-interest as well as our custom profile.

In Figure 17, we show a notional attack of a the ISR mission through the Global Positioning System (GPS) receiver of the Autopilot (which is in turn a part of the Platform). Of interest here, each of the non-quoted terms is actually a strongly identified reference of another part of the system: part = gps means "the part gps of a usage of Autopilot can be replaced with this Compromised GPS Receiver" while addsGeneralization = Misinformation and toFlow = Information are references to specific parts of the model.

While this diagram lacks the overall communication expressiveness of the baseline system, it succinctly captures the changes to the assumptions in evaluating the context.

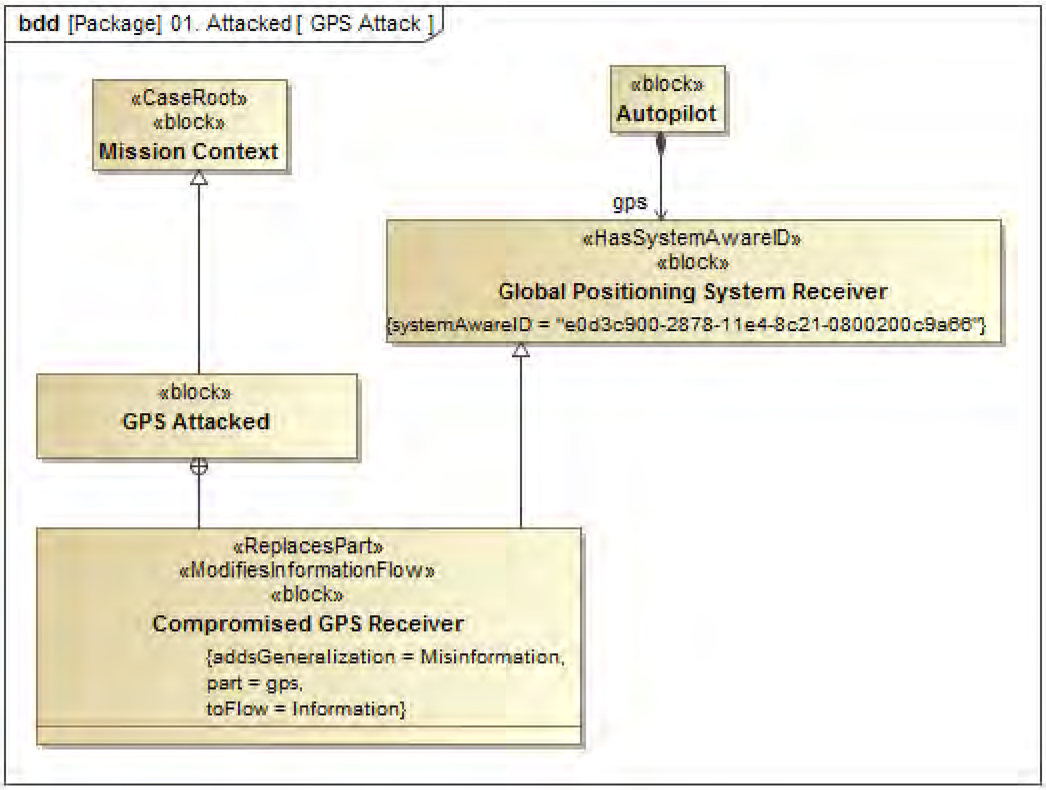

bdd [Package] 01. Attacked [GPS Attack]
«CaseRoot»
«block»
Mission Context
«block»
Autopilot
gps
«HasSystemAwareID»
«block»
Global Positioning System Receiver
{systemAwareID = "e0d3c900-2878-11e4-8c21-0800200c9a66"}
«block»
GPS Attacked
«ReplacesPart»
«ModifiesInformationFlow»
«block»
Compromised GPS Receiver
{addsGeneralization = Misinformation,
part = gps,
toFlow = Information}

Figure 17 - SysML Block Definition Diagram of an attacked ISR Mission

Case Study: Modeling a Change of Understanding to a Mission

In its role as a living artifact, tying the overall knowledge of a mission together, the system model will change over time. While changes to the underlying model of the system-of-interest may be desirable, certain changes may be better reflected as deltas to the system, specifically if knowledge of this change is not generally available, proposed or otherwise not yet implemented. Figure 18 shows such an evolution, where the specific Payload on the platform is extended to be a specific type of sensor.

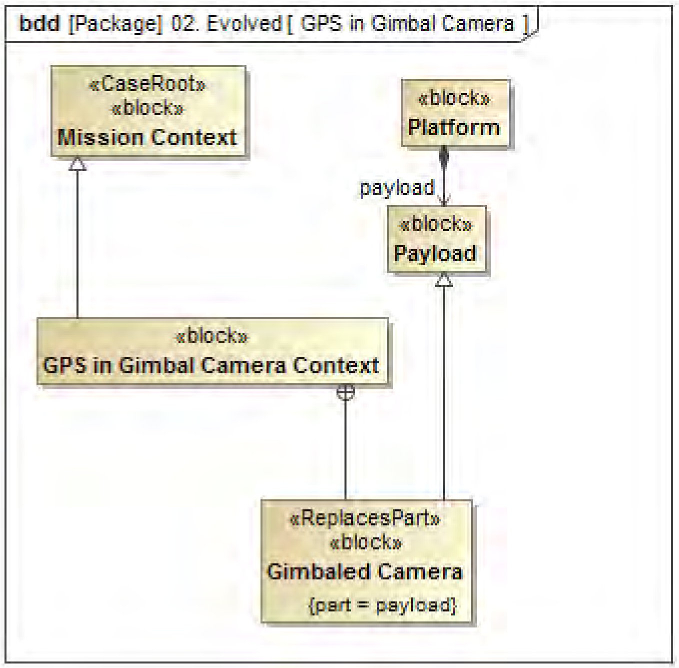


Figure 18 - SysML Block Diagram of a More Detailed understanding of the ISR Mission

7 Case Study: Modeling Defense of a Mission

Composing both the attack and a changed understanding, it becomes possible to model a new state of the mission that can be evaluated to reflect a change to the systems performing the mission to yield a more resilient overall mission. In Figure 19, a System-Aware sentinel is introduced somewhere in the system.

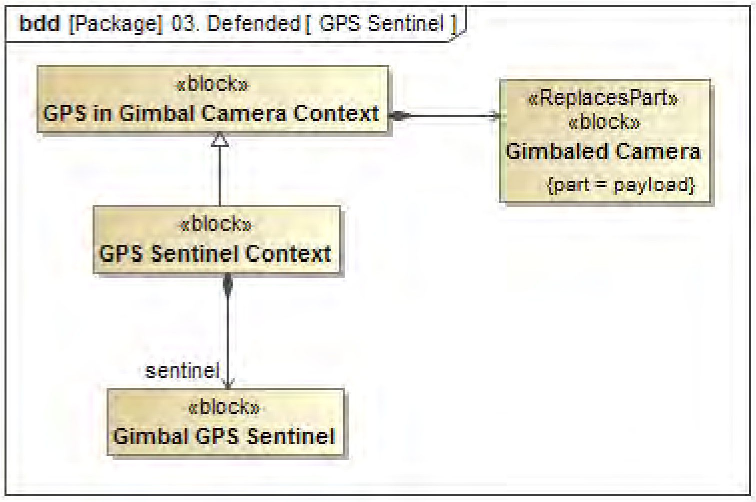


Figure 19 - SysML Block Definition Diagram of a System-Aware defended ISR Mission

In order to reflect the new, less-naive failure mode of the system, Figure 20 shows a derived requirement of the system, along with its traceability to a known attack and a proposed system, which can satisfy this requirement.

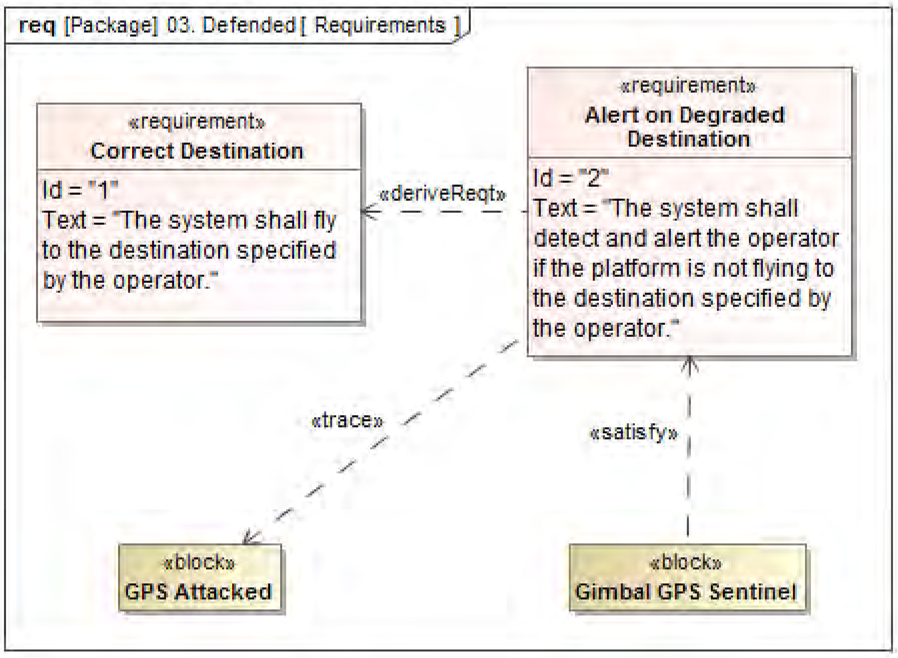

req [Package] 03. Defended [Requirements]
«requirement»
Correct Destination
Id = "1"
Text = "The system shall fly to the destination specified by the operator."
«deriveReqt»
«requirement»
Alert on Degraded Destination
Id = "2"
Text = "The system shall detect and alert the operator if the platform is not flying to the destination specified by the operator."
«trace»
«satisfy»
«block»
GPS Attacked
«block»
Gimbal GPS Sentinel

Figure 20 - SysML Requirements Diagram of a System-Aware defended ISR Mission

8 Case Study of SysML Informed Attack Tree: UAV Walk-off Attack

As stated previously, a simplified Intelligence, Surveillance and Reconnaissance (ISR) mission is the case study example for the UAV system. One of the onboard UAV sensor systems that have far reaching implications with respect to UAV flight control, navigation, and mission operations is the Global Position System (GPS). GPS is used systemically in UAV many operations, from flight path planning, waypoint navigations, blue force tracker, and vehicle control to mission-support systems like image and video acquisition. Because of its pervasive use in UAV operations and mission analysis activities we justify the need to access the impact of compromised GPS devices may have on downstream systems that depend on GPS information for their operations.

All GPS receivers work on the same basic principles to calculate a navigation solution in 3D space and time. The navigation solution is calculated by trilateration where the receiver measures its distance from four (or more) satellites: one to resolve each dimension in space-time. Each satellite generates and broadcasts a unique, public pseudo-random number (PRN) stream called the coarse acquisition (C/A) code, which repeats every 1ms. The current time, as determined by an atomic clock on each satellite, week number, and other navigation information is modulated as a navigation message on top of the C/A code. GPS receivers generate their own local replica of each satellite's C/A code and estimate the time delta required to align the local replica to the received copy. The time delta, along with the transmission times of each C/A code signal form the distance measurements called pseudo-ranges. The receiver also decodes the navigation data in order to calculate the satellites' positions and clock offsets. All this information is used to accurately estimate the 3D position and time.

Much of the research in the open literature has focused on two attacks against GPS: jamming and spoofing. Jamming simply transmits noise in the GPS frequency band, preventing a receiver from locking onto the GPS signal. Spoofing attacks are a very specific type of attack that forges the information used to calculate pseudo-ranges. These attacks are not on the receiver itself per se, as the receiver operates properly just with bogus input data. At a high level, previous work was limited to showing that receivers when given bogus data will output a bogus navigation solution.

Previous research in the literature has not examined in a detailed manner how actual receivers may be compromised *in-situ* by insider threats or propagated malware and how corrupted GPS information flows down to dependent systems to potentially affect many downstream UAV systems. Previous efforts in Phase 1 looked modestly at the potential attack surface of GPS with respect to UAV operations. Our goal in this effort was to see how SysML models could better inform us on potential GPS attacks for selected UAV systems, how those attacks could be better understood in the context of SysML mission perspective, and finally how we could develop integrated design solutions for such attacks.

To understand the attack surface associated with GPS functions, we refer to figure 20 to show how GPS is used in typical UAV operations. GPS sensory data in the autopilot is forwarded directly to two systems, thus it directly influences these systems. These systems are INS/State Estimator, and Flight Controller. The INS (gyros, and accelerometers) uses the GPS location fixes to correct residual errors in the Inertial Measurement Unit (IMU) of the INS. This update occurs on regular interval of anywhere from 10 seconds to a 10s of minutes. The GPS main function in the autopilot is to forward lat/long coordinates to the three loops in the Fight controller; mission planning loop, heading and attitude loop, and altitude loop. The mission-planning loop is part of the high-level autopilot functionality, it provides tracking capability with respect to stored waypoint information. The GPS provides updates on vehicle geo-reference position to the mission planner, which compares GPS geo-reference updates to the stored waypoints. The mission planner then commands the controller to “fly” the UAV in the direction of the next waypoint.

In Figure 21 below, the blocks circled in red are systems that are directly or indirectly influenced by GPS data. We can see from Figure 21, that the extent of influence of a compromised GPS receiver reaches the terminal point of the flight control surfaces. While we can see the high level relationships at this model level, we would to use the power of the model to see all potential relationships any function has with GPS. The GPS attacks that we are interested in exploring with SysML assistance are attacks that are; (1) can be embedded into the autopilot system, (2) do not necessarily require an external trigger to active, and (3) are stealthy. We can do that by exploiting the querying capability of our SysML tool and models.

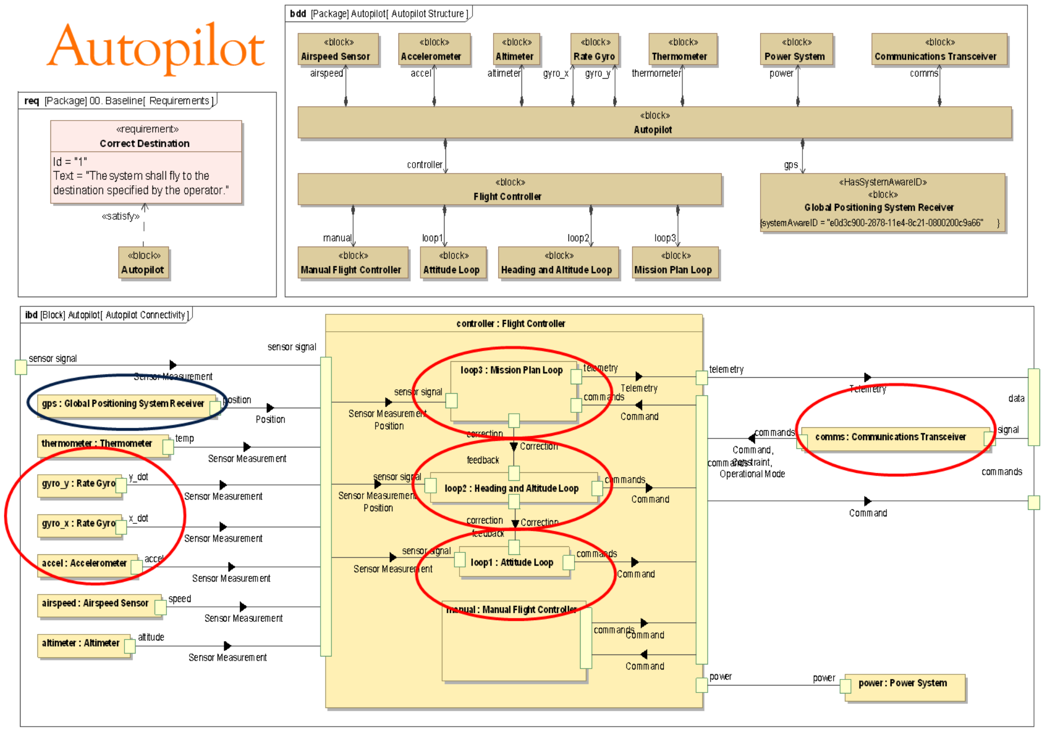
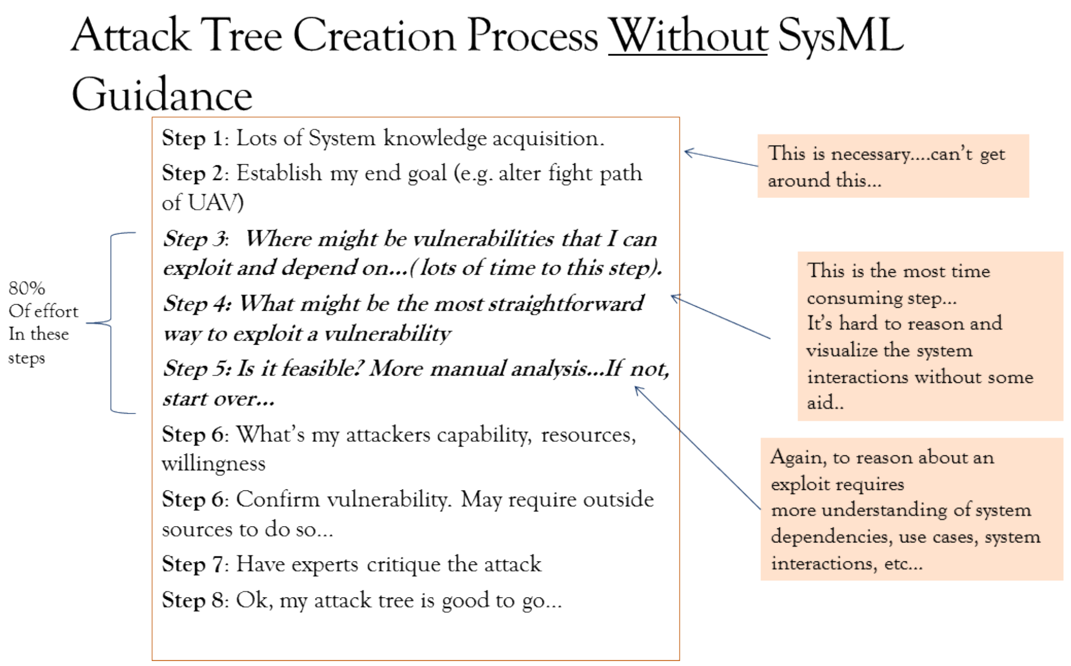

Autopilot
req [Package] 00. Baseline[Requirements]
«requirement» Correct Destination
Id = "1"
Text = "The system shall fly to the destination specified by the operator."
«satisfy»
«block» Autopilot
bdd [Package] Autopilot[Autopilot Structure]
«block» Airspeed Sensor
«block» Accelerometer
«block» Altimeter
«block» Rate Gyro
«block» Thermometer
«block» Power System
«block» Communications Transceiver
«block» Autopilot
«block» Flight Controller
«HasSystemAwareID» «block» Global Positioning System Receiver
«block» Manual Flight Controller
«block» Attitude Loop
«block» Heading and Altitude Loop
«block» Mission Plan Loop
ibd [Block] Autopilot[Autopilot Connectivity]
controller : Flight Controller
gps : Global Positioning System Receiver
thermometer : Thermometer
gyro_y : Rate Gyro
gyro_x : Rate Gyro
accel : Accelerometer
airspeed : Airspeed Sensor
altimeter : Altimeter
loop3 : Mission Plan Loop
loop2 : Heading and Altitude Loop
loop1 : Attitude Loop
manual : Manual Flight Controller
comms : Communications Transceiver
power : Power System
Sensor Measurement
Position
Telemetry
Command
Correction
feedback

Figure 21 - SysML Model for the Autopilot

It is worthwhile to talk briefly about the process of creating attack trees without any notational model based assistance. As we mentioned in previous section, attack tree's creation relies on Subject Matter Experts to help create both the structure of the attack tree (logical steps in the attack) and the threat agent profile. Of these two, the structure of the attack tree depends heavily on the details of the system-to-system relationships, interactions, and how they relate to users and missions. Referring to Figure 22 below, we can distill the process of creating attack trees in 8 steps. Step 1 is always present for both model assisted and non-model assisted creation processes. Step 2 is defining the objectives on an attack or a class of attacks. The real work of creating realistic and credible attack trees is step 3 through 5. This is the process of discovery, system exploration, and entry points into the system from an attacker's perspective. These steps are difficult to successfully and completely achieve without some means to see how all of the functions/components of the system interact in both nominal and off-nominal uses. When these steps are executed manually by engineers, analysts, and subject matter experts, approximately 80% of the overall effort to create attack trees is spent through consensus based engineering trying to "connect the dots". Furthermore, we have observed from our experiences and confirmed by others, that attack trees without model based assistance have a tendency to fall back to "text-book" vulnerabilities, instead of system specific vulnerabilities. By "text-book" we mean they were mostly technical and generic in nature, and did not account for different users of the system, mission or organizational policy vulnerabilities.



Attack Tree Creation Process Without SysML Guidance

Step 1: Lots of System knowledge acquisition.

Step 2: Establish my end goal (e.g. alter fight path of UAV)

***Step 3**: Where might be vulnerabilities that I can exploit and depend on…(lots of time to this step).*

***Step 4**: What might be the most straightforward way to exploit a vulnerability*

***Step 5**: Is it feasible? More manual analysis…If not, start over…*

Step 6: What's my attackers capability, resources, willingness

Step 6: Confirm vulnerability. May require outside sources to do so…

Step 7: Have experts critique the attack

Step 8: Ok, my attack tree is good to go…

80% Of effort In these steps

This is necessary….can't get around this…

This is the most time consuming step… It's hard to reason and visualize the system interactions without some aid..

Again, to reason about an exploit requires more understanding of system dependencies, use cases, system interactions, etc…

Figure 22 - Attack Tree Creation Process without SysML

9 Attack Tree Modeling Process with SysML Models

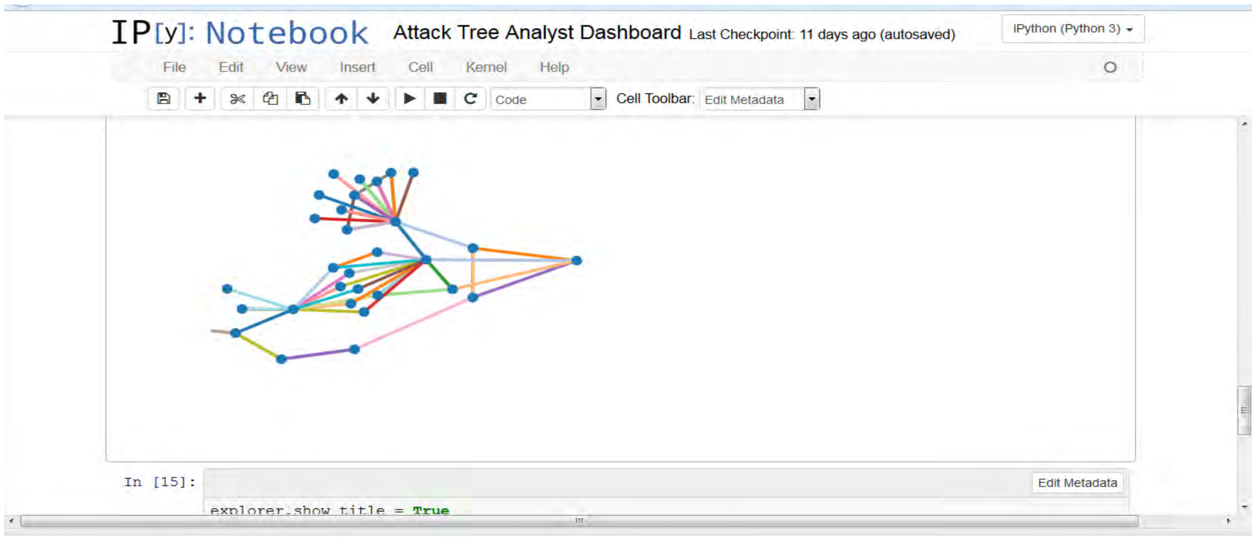
SysML models can be used to inform attack tree modeling, more specifically, the models can used to explore the *attack surface* to find vulnerabilities. We can define *attack surface* as a systems exposure, to a set of reachable and exploitable vulnerabilities that a system's resources has to a class of adversarial threat actors. A *surface vulnerability* is a path to set of systems resources. The *exploitation* of those resources through the vulnerability by an attacker creates the opportunity to spread the effects of exploitation to other functions and users of the system. At the broadest stance, when trying to characterize an attack surface for a given system, we typically look at interrelated aspects that define how the system operates, is used and interacts with the world. These typically include things like:

- Attack origins - development, local, remote, maintenance, etc..
- Propagation - Software resources, insider, networks, social engineering, etc...
- Surface Vulnerability Opportunities - modifiable configuration, unintended side channels, alleviated privilege, intercepting data, spoofing data or commands, etc.
- System Service Effects - What effects are possible to degrade or alter system service.

As expected the process of creating an "SysML driven" attack tree is different than the manual process of creating attack trees. Figure XX below illustrates the SysML driven process. As before, threat actor profiling is done entirely in the Secure*IT*ree Attack tree framework. In SysML model based environment, knowledge acquisition produces specifications for creating models of the target system at various abstraction levels. In the System Aware framework, mission context inherits models at the architecture, platform, and functional levels. This feature allows us to explore attack surfaces and vulnerabilities with respect to the mission scenarios. At this stage in our work, our primary tool for exploring attack surfaces is relationships between functions, platforms, architectures and missions via path analysis. By finding relational paths between functions, components we can identify places in the attack surface where; (1) a vulnerability would cause concern, (2) paths are overlooked paths, and (3) potential backdoors. Once a relational map of paths is extracted and created from the models, we use this information to create different attack scenarios or paths in the attack tree. The paths extracted from SysML model are isomorphic to the paths in an attack tree. Once the attack trees are instantiated with the information collected from the path analysis, we can build the attack tree and perform quantitative analysis is on it to solve metrics like ease of attack or desirability for a given path. These metrics enable decision making/tradeoff analysis on type and placement of cyber defenses as constrained by costs and mission criticality. The SysML model/attack tree interaction is a process of refinement until architectural remediation is sufficient for the application and mission. Cyber defense are evaluated in the Attack Tree as *countermeasure trees*, and the entire attack tree is solved again to see if the metrics ease of attack, desirability metrics have significantly changed in the favor of the defender.

10 Querying Models to Explore Potential Vulnerability Space

To promote broader use and portability among different SysML authoring tools, we chose to build a "Attack Tree Analyst DashBoard" in a open source iPython environment that allows the system modeler, attack tree analyst, and manager to work collaboratively toward vulnerability exploration of the system model. At present our iPython dashboard has features for functional relationship exploration only. We use the IPython dashboard to provide a visual representation of "source to target" information flow. Source is where information originates from, and target is the "sink" – where the information is ultimately consumed. Figure 23 is a snapshot of the dashboard displaying a relationship graph of various functions in the autopilot.


IP[y]: Notebook
Attack Tree Analyst Dashboard Last Checkpoint: 11 days ago (autosaved)
IPython (Python 3)
File
Edit
View
Insert
Cell
Kernel
Help
Code
Cell Toolbar: Edit Metadata
Edit Metadata

```
In [15]:
explorer.show_title = True
```

Figure 23 - Relationship Diagrams in the Autopilot

11 Formulating Path Graphs from Attack Tree Dashboard

Given the goal of finding potential vulnerabilities in the attack surface that could lead to a GPS walk-off attack, we use the iPython dashboard to search for paths that are sourced from GPS and terminate at other functions. Figure 24 below shows the results of one such query search.

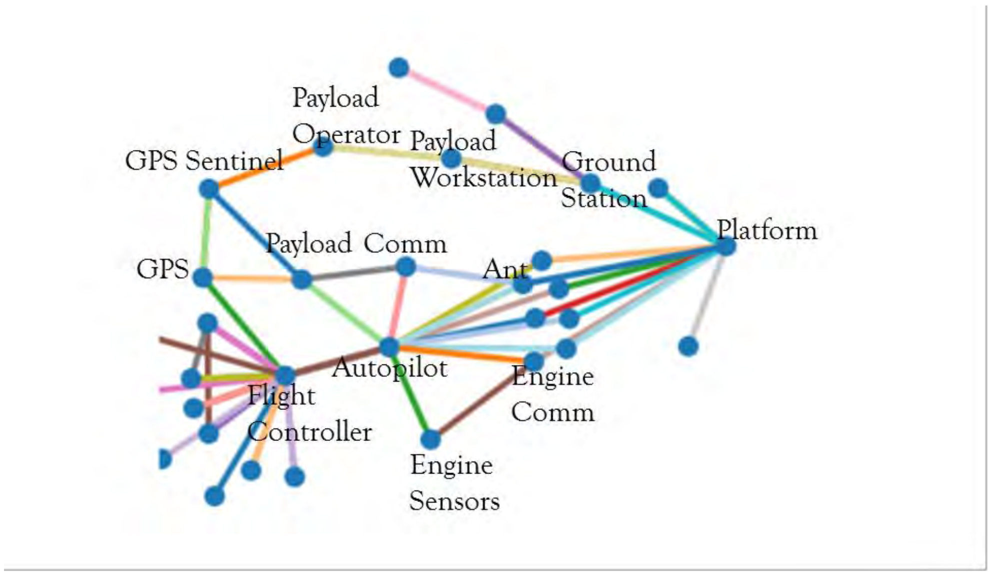


Figure 24 - Query Seach Results

The dashboard produces a list of paths from source to sink based on length of path

- (2) GPS -->Flight controller
- (3) GPS-->Payload-->Autopilot
- (3) GPS-->Flight Controller-->Autopilot
- (4) GPS-->GPS Sent-->Payload-->Autopilot
- etc…..

Based on path analysis alone we can sort paths according to length or unusual traversal or connectedness.

Most Direct Paths (shortest path):
•GPS--> Flight controller
•GPS--> Payload-->Autopilot

Intermediate Paths:
•GPS-->Flight Controller-->Autopilot
•GPS-->Payload-->Comm-->Autopilot

•Some Unusual Paths:
•Ground-Station -->Platform-->Engine Comm-->Autopilot
•Platform-->Comm-->Autopilot
These paths don't directly have GPS as source, but GPS is connected to the autopilot.

Focusing on the GPS walk-off attack paths we see that GPS directly affects 4 downstream systems and their internal functions. These systems are Autopilot, flight controller, payload, and actuation. Shortest paths are often first choices in search for vulnerabilities, but not always productive because defenders may have anticipated this placed defenses there. Unusual paths are often "backdoors", possibly not defended well, but can be more complicated to exploit. Long paths are often multi-step which suggest several coordinated actions to achieve the attack.

Let's explore one of the paths in our list. GPS-->Flight Controller-->Autopilot. In Figure 25 below we have a block diagram of a typical adafruit GPS receiver. If we were modeling this in SysML as functional block we would only be interested in the functionality at the digital interfaces and components of the GPS receiver. Referring to Figure 25, we can see there are several digital interfaces to the chip. These may places to intercept and modify information, inject malware, or upload different configuration profiles into the chip.

Constructing a scenario around GPS-->Flight Controller-->Autopilot path yields the following plausible exploit.

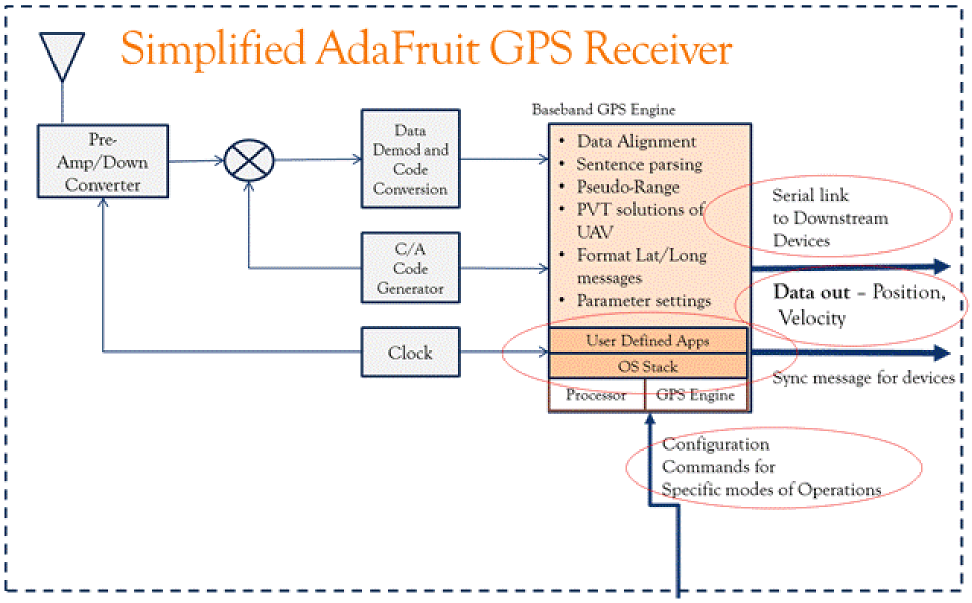


Figure 25 - Block Diagram of a Typical AdaFruit GPS Receiver

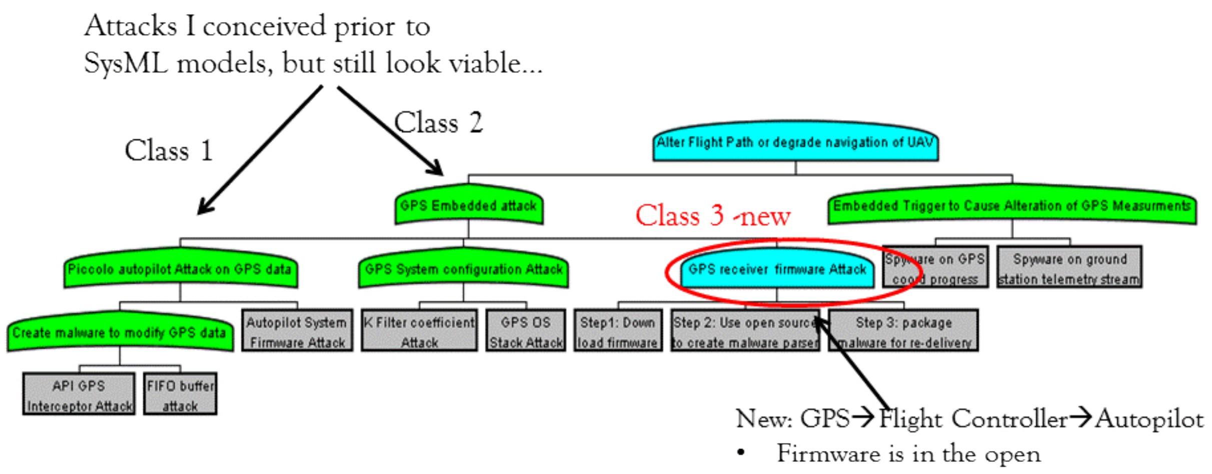
So one scenario is that GPS is compromised at the firmware level. By firmware we mean that the code that runs on the GPS co-processor is altered and reloaded on the GPS chip. Is this plausible? As it turns out, AdaFruit GPS Firmware (as binary) is open and available on company GIT site. Firmware upgrades occur from time to time, and vendors may want to upgrade new features. This still leaves the attacker with binary firmware, but reverse engineering tools like IDApro can assist in finding system calls relating to GPS sentences. we can then use the AdaFruit client data handler package to build a lightweight parser to parse out sentences in the background and watch for zero day "position coordinates". Once zero-day is reached, calculate GPS sentence off-sets or do a look up table. Merge real GPS plus zero-day Off-sets and send through the payload processor (it's a pass-through) to the Autopilot... Were done for this scenario. Below in Figure 26 is actual snippet of code for altering GPS sentences.

```
........
}
int32_t degree;
long minutes;
char degreebuff[10];
// look for a few common sentences
if (strstr(nmea, "$GPGGA")) {
// found GGA
char *p = nmea;
// get time
p = strchr(p, ',')+1;
float timef = atof(p);
uint32_t time = timef;
hour = time / 10000;
minute = (time % 10000) / 100;
seconds = (time % 100);
milliseconds = fmod(timef, 1.0) * 1000;
// parse out latitude......
```

Figure 26 - GPS Code Snippet

As we can see this seems plausible, but by whom? This was probably not a teen hacker, but certainly within the realm of possibilities for a nation state or cyber-criminal group. Once the scenario is discussed in a consensus meeting with analysts, system engineers, and managers, decisions can be made as what to do to remedy the issue or what not to do. To aid in decision-making we can insert the scenario into an existing attack tree of the system or build attack tree from scratch. Figure 27 is an attack tree where we inserted the new path.

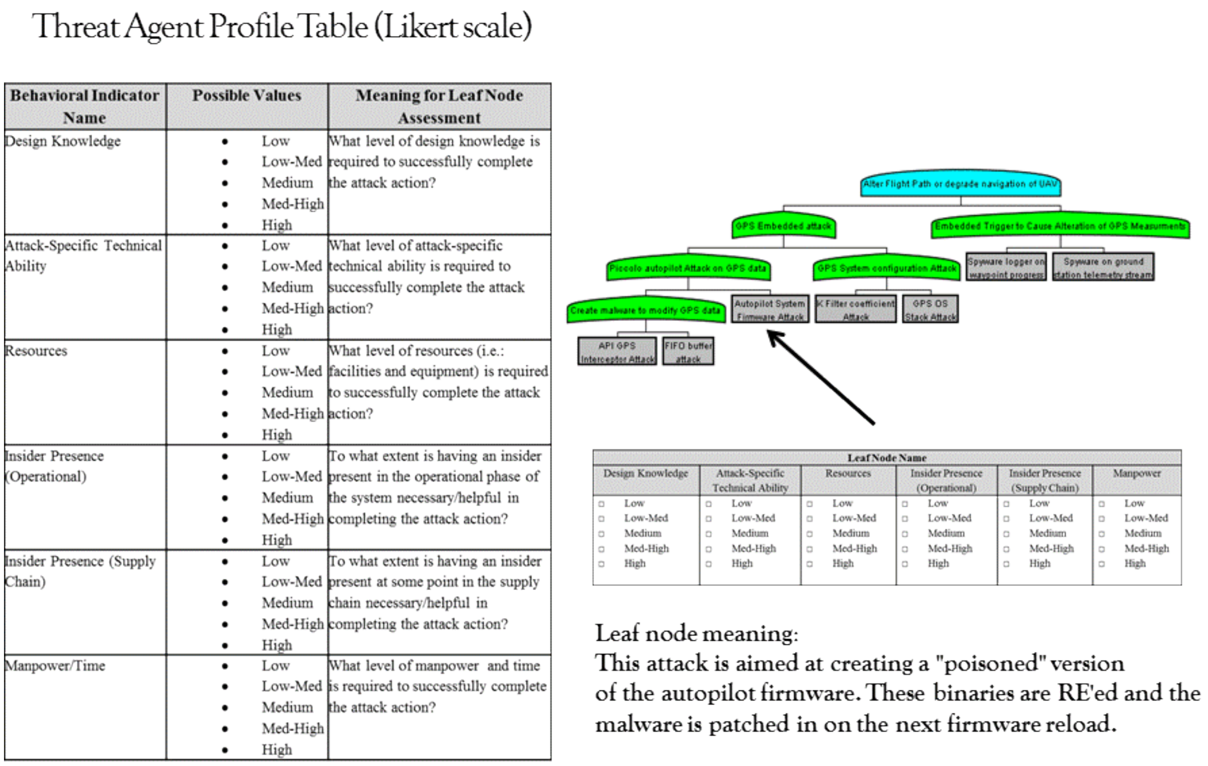
Augmented Attack Tree for Newly Found GPS Attacks


Attacks I conceived prior to SysML models, but still look viable...
Class 1
Class 2
Alter Flight Path or degrade navigation of UAV
GPS Embedded attack
Class 3 -new
Embedded Trigger to Cause Alteration of GPS Measurments
Piccolo autopilot Attack on GPS data
GPS System configuration Attack
GPS receiver firmware Attack
Spyware on GPS coord progress
Spyware on ground station telemetry stream
Create malware to modify GPS data
Autopilot System Firmware Attack
K Filter coefficient Attack
GPS OS Stack Attack
Step1: Down load firmware
Step 2: Use open source to create malware parser
Step 3: package malware for re-delivery
API GPS Interceptor Attack
FIFO buffer attack
New: GPS→Flight Controller→Autopilot
Firmware is in the open

Using a model driven approach, we uncovered additional vulnerabilities and the exploit to realize the attack.

Figure 27 - Attack Tree with New Path Inserted

Now that we have inserted a new path in the attack tree, we have to run the threat agent profile against the tree to determine the ease of attack. The agent profiles for this attack tree used utility functions based on Likert scales shown below in Figure 28. Each leaf node is defined by series of attributes, namely, design knowledge, attack specific capability, resources, manpower and insider presence.


Alter Flight Path or degrade navigation of UAV
GPS Embedded attack
Embedded Trigger to Cause Alteration of GPS Measurments
Piccolo autopilot Attack on GPS data
GPS System configuration Attack
Spyware logger on waypoint progress
Spyware on ground station telemetry stream
Create malware to modify GPS data
Autopilot System Firmware Attack
K Filter coefficient Attack
GPS OS Stack Attack
API GPS Interceptor Attack
FIFO buffer attack

Threat Agent Profile Table (Likert scale)

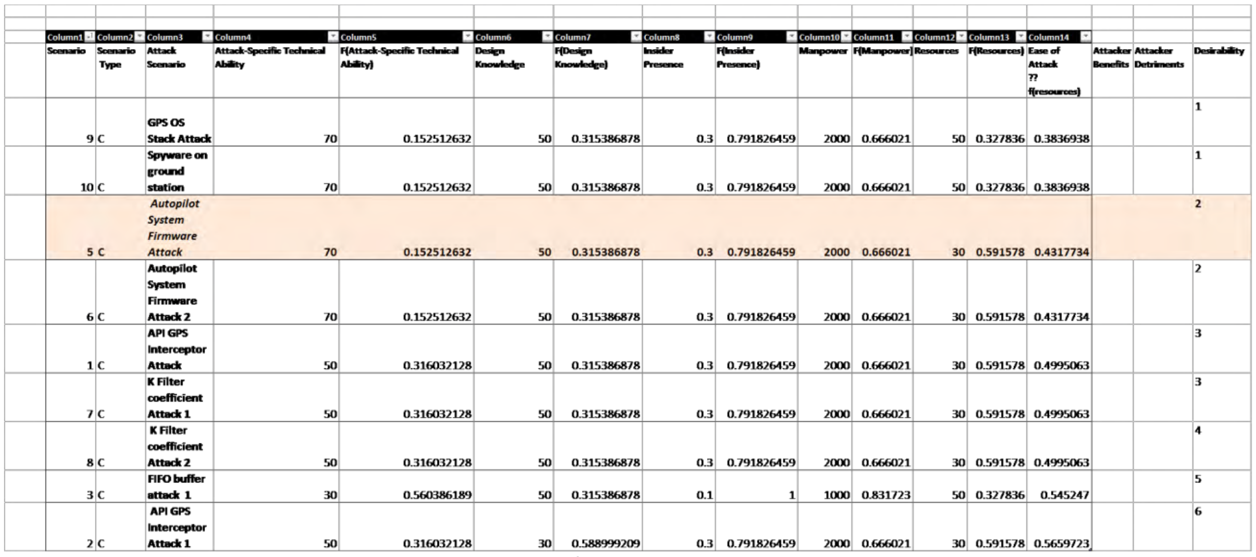
Behavioral Indicator Name	Possible Values	Meaning for Leaf Node Assessment
Design Knowledge	• Low • Low-Med • Medium • Med-High • High	What level of design knowledge is required to successfully complete the attack action?
Attack-Specific Technical Ability	• Low • Low-Med • Medium • Med-High • High	What level of attack-specific technical ability is required to successfully complete the attack action?
Resources	• Low • Low-Med • Medium • Med-High • High	What level of resources (i.e.: facilities and equipment) is required to successfully complete the attack action?
Insider Presence (Operational)	• Low • Low-Med • Medium • Med-High • High	To what extent is having an insider present in the operational phase of the system necessary/helpful in completing the attack action?
Insider Presence (Supply Chain)	• Low • Low-Med • Medium • Med-High • High	To what extent is having an insider present at some point in the supply chain necessary/helpful in completing the attack action?
Manpower/Time	• Low • Low-Med • Medium • Med-High • High	What level of manpower and time is required to successfully complete the attack action?

Leaf Node Name					
Design Knowledge	Attack-Specific Technical Ability	Resources	Insider Presence (Operational)	Insider Presence (Supply Chain)	Manpower
☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High	☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High	☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High	☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High	☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High	☐ Low ☐ Low-Med ☐ Medium ☐ Med-High ☐ High

Leaf node meaning:
This attack is aimed at creating a "poisoned" version of the autopilot firmware. These binaries are RE'ed and the malware is patched in on the next firmware reload.

Figure 28 - Attack Tree with Attack Profiles

The tree is now ready to calculate for ease of attack and other metrics. Table 2 shows the results of calculating all the scenarios through the attack tree. Scenario 5 is the new attack path and the ease of attack is .431 and is ranked 2nd in desirability. Given the low ease of attack and the desirability factor for a nation state actor, this would indicate further action be taken in the way of cyber defenses if the system were exposed to nation actors.



Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14			
Scenario	**Scenario Type**	**Attack Scenario**	**Attack-Specific Technical Ability**	**F(Attack-Specific Technical Ability)**	**Design Knowledge**	**F(Design Knowledge)**	**Insider Presence**	**F(Insider Presence)**	**Manpower**	**F(Manpower)**	**Resources**	**F(Resources)**	**Ease of Attack ?? f(resources)**	**Attacker Benefits**	**Attacker Detriments**	**Desirability**
9	C	**GPS OS Stack Attack**	70	0.152512632	50	0.315386878	0.3	0.791826459	2000	0.666021	50	0.327836	0.3836938			1
10	C	**Spyware on ground station**	70	0.152512632	50	0.315386878	0.3	0.791826459	2000	0.666021	50	0.327836	0.3836938			1
5	C	*Autopilot System Firmware Attack*	70	0.152512632	50	0.315386878	0.3	0.791826459	2000	0.666021	30	0.591578	0.4317734			2
6	C	**Autopilot System Firmware Attack 2**	70	0.152512632	50	0.315386878	0.3	0.791826459	2000	0.666021	30	0.591578	0.4317734			2
1	C	**API GPS Interceptor Attack**	50	0.316032128	50	0.315386878	0.3	0.791826459	2000	0.666021	30	0.591578	0.4995063			3
7	C	**K Filter coefficient Attack 1**	50	0.316032128	50	0.315386878	0.3	0.791826459	2000	0.666021	30	0.591578	0.4995063			3
8	C	**K Filter coefficient Attack 2**	50	0.316032128	50	0.315386878	0.3	0.791826459	2000	0.666021	30	0.591578	0.4995063			4
3	C	**FIFO buffer attack 1**	30	0.560386189	50	0.315386878	0.1	1	1000	0.831723	50	0.327836	0.545247			5
2	C	**API GPS Interceptor Attack 1**	50	0.316032128	30	0.588999209	0.3	0.791826459	2000	0.666021	30	0.591578	0.5659723			6

Table 2 - Results of Attack Pathway Calculations

12 Findings and Lessons learned

A final observation we note is the experience of using a systematic model driven process to conduct cyber-vulnerability analysis often yields more information than just quantifying the vulnerability aspects of the system. The process itself is a learning experience, providing a richer insight into how a system behaves in response to potential threats and attacks. The inclusion of this information into review processes and cyber procurement activities can only enlighten how we manage cyber defense capabilities from a mission perspective. Finally, the process of conducting studies on how to integrate SysML models into Attack Tree disciplines allows two very important pieces of information to come into direct connection with each other: What the system is supposed to do, and what it actually does. This information is essential for cyber-defense V&V activities and reviews.

13 Planned Activities for 2015

The Phase 2 activity led to substantial progress in conceptualizing the functions of model-based tools and techniques that could provide significant support to System-Aware Architectural Selection and Assessment. The Naval Fleet Cyber Command workshop provided an independent assessment relate to the opportunity for using model-based tools in the cybersecurity domain. However the hypothesis that these tools yield a truly scalable approach remains largely untested. Future efforts should focus on the following items:

- Continued development of integration between systems and attack tree models. This would include (1) the definition of SysML design patterns that show how a countermeasure is added and how the corresponding countermeasure tree is built in SecureITree and (2) continued definition of a notional workflow for the multiple teams engaging in the solution-selection collaborative environment, building on Workbench concepts.
- Extension of the six-step methodology discussed in this report, to support application in hierarchical and systems-of-systems settings.

- Application of the methods in hierarchical and systems-of-systems case studies to assess the overall effort involved in the selection of the defensive architecture and the degree to which the concept can support an agile environment through the reuse or combination of architectures.

The hypotheses of the proposed research effort are that 1) The Synergistic use of advanced tools can provide greater assurance that cyber-attack analysis and defensive solution selection will be comprehensive and, as a result, greatly enhanced, and 2) that the tools can be embedded into decision-processes in a practical manner to facilitate “what-if” analysis. These hypotheses will be addressed through experimentation in the following manner:

- We will select an application system as a first example of a system to be considered for analysis, with considerations that the selected example system is sufficiently complex to serve as a useful first step, but not so complex that the time allotted for this initial phase is insufficient to conduct the needed experiments and analyses.
- We will develop the needed tool Integration Principles and Information exchange requirements on top of the selected baseline tools—MagicDraw and SecureITree—to support multi-criteria decision-making that will be evaluated in the context of the selected application.
- We will carry out a multi-criteria analysis using the tools, and we will provide expert assessment about the quality of analysis enabled by the tools and the related impact on implementation decisions.
- We will conduct a workshop with interested potential users to expose the results of our effort and to identify the needed activities to bring the modeling support into actual use for decision support.

14 References

[1] Jonathan D. Weiss. A system security engineering process. In 14th Nat. Comp. Sec. Conf., pages 572–581, 1991

[2] Edward G. Amoroso. Fundamentals of Computer Security Technology. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[3] Terrance R. Ingoldsby. Understanding Risk Through Attack Tree Analysis. Computer Security Journal, 20(2):33–59, 2004.

[4] Poramate Ongsakorn, Kyle Turney, Mitchell A. Thornton, Suku Nair, Stephen A. Szygenda, and Theodore Manikas. Cyber threat trees for large system threat cataloging and analysis. In 4th Annual IEEE Systems Conference, pages 610–615, April 2010.

[5] Inger Anne Tøndel, Jostein Jensen, and Lillian Røstad. Combining Misuse Cases with Attack Trees and Security Activity Models. In International Conference on Availability, Reliability and Security, pages 438–445, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[6] Terry Tidwell, Ryan Larson, Kenneth Fitch, and John Hale. Modeling Internet Attacks. In Proceedings of the 2nd IEEE Systems, Man and Cybernetics Information Assurance Workshop (IAW '01), pages 54–59, West Point, USA, June 2001.

[7] Indrajit Ray and Nayot Poolsapassit. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In Sabrina di Vimercati, Paul Syverson, and Dieter Gollmann, editors, ESORICS'2005, volume 3679 of LNCS, pages 231–246. Springer Berlin / Heidelberg, 2005.

[8] Darrell M. Kienzle and William A. Wulf. A Practical Approach to Security Assessment. In Proceedings of the 1997 New Security Paradigms Workshop, NSPW '97, pages 5–16, New York, NY, USA, 1997. ACM.

[9] John N. Whitley, Raphael C.-W. Phan, JieWang, and David J. Parish. Attribution of attack trees. Computers & Electrical Engineering, 37(4):624–628, 2011.

[10] Bruce Schneier. Attack Trees: Modeling Security Threats. Dr. Dobb's Journal of Software Tools, 24(12):21–29, 1999.

[11] Stefano Bistarelli, Pamela Peretti, and Irina Trubitsyna. Analyzing Security Scenarios Using Defence Trees and Answer Set Programming. Electron. Notes Theor. Comput. Sci., 197(2):121–129, 2008.

[12] Ahto Buldas and Roman Stepanenko. Upper Bounds for Adversaries' Utility in Attack Trees. In Jens Grossklags and Jean C. Walrand, editors, GameSec, volume 7638 of LNCS, pages 98–117. Springer, 2012.

[13] L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modeling Language, Pearson Publishers, 2013.