Security Engineering Project:

System Aware Cyber Security for an Autonomous Surveillance System On Board an Unmanned Arial Vehicle

Technical Report SERC-2014-TR-036-3

January 31, 2014

Principal Investigator: Dr. Barry Horowitz, University of Virginia

Co-Principal Investigators:

Georgia Tech Research Institute: Dr. William Melvin,

University of Virginia: Dr. Peter Beling, Dr. Kevin Skadron,

Dr. Ron D. Williams

Contract Number: H98230-08-D-0171 Task Order: 0104, RT 042

Report No. SERC-2014-TR-036-3

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract H98230-08-D-0171 (Task Order 0104, RT 042).

Executive Summary

The Systems Engineering Research Center (SERC) has developed a novel cyber security concept for embedding security solutions into systems called *System-Aware cyber security*. The 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) 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.

To demonstrate the effectiveness of the System-Aware design patterns, specific ones were developed for an unmanned aerial vehicle (UAV) application. 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.

During the Phase I effort the UVa/GTRI team achieved a number of accomplishments including:

- Creation of software and hardware in the loop simulators and emulators to enable the testing of System-Aware cyber security solutions.
- Identification, selection, and design of potential cyber-attacks that could be utilized to compromise the UAV's ability to carry out its mission.
- Design of a prototype smart security Sentinel to host System-Aware cyber security solutions to protect against the cyber-attacks.
- Design of a Sentinel for airborne use based on the SiCore SHIELD secure single board computer.

The Phase II effort for conducting a flight demonstration of the System-Aware Sentinel has been planned. Phase II will consist of the activities necessary to integrate the results of the Phase I effort into the GTRI Aerial Unmanned Sensor System (GAUSS) aircraft in order to create a flight-ready demonstration. The GAUSS platform is a small research UAV with a widely used, commercial off-the-shelf autopilot system and camera gimbal. The demonstration will show how the System-Aware approach can be used to thwart cyber-attacks against autopilot systems and sensor systems.

Table of Contents

List of Figures

List of Tables

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 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.

In addition, a Phase 0 effort consisting of an evaluation of possible applications of existing RT-28-developed design patterns 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 prototyping pilot effort for validating the System-Aware cyber security 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.

This document outlines the results of the Phase 1 effort to apply System-Aware cyber security solutions to an UAV system:

1. Creation of both a simulated and an emulated environment to enable the testing of System-Aware cyber security solutions.
2. Identification, selection, and design of potential cyber-attacks that could be utilized to compromise the UAV's ability to carry out its mission.
3. Design of a prototype smart security Sentinel to host System-Aware cyber security solutions to protect against the cyber-attacks identified in (2).

4. Design for a Sentinel configured to meet the size, weight, power and functional requirements necessary for airborne use.

This document also outlines the proposed work for a follow-on Phase 2 project that would consist of activities necessary to integrate the results of the Phase 0 and 1 efforts into the GTRI Aerial Unmanned Sensor System (GAUSS) aircraft in order to create a flight-ready demonstration.

2 Project Emulation and Simulation 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. 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. The following subsections describe the SiL and HiL development and test environment at each location.

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 done by using the Piccolo II, operator interface for piloting 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.

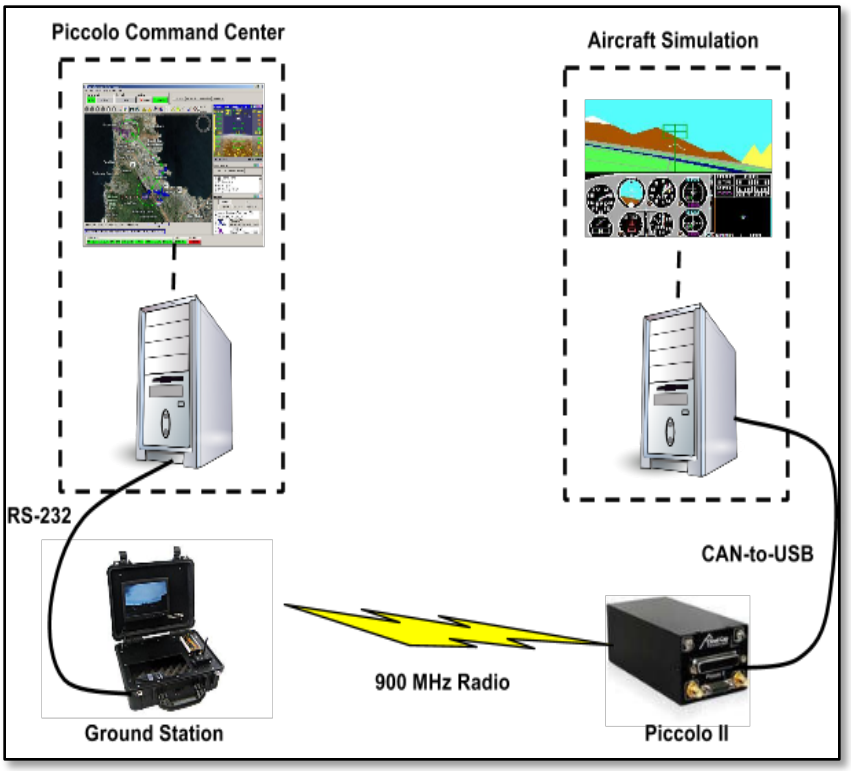


Figure 1. Basic HiL configuration for the Piccolo II.

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. The details of how the HiL emulation environment was augmented to include the Sentinel's monitoring, detection, and restoration techniques is described in Section 3.1.

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 200 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 200 gimbal, Cloud Cap video processing system (VPS), and ground station. The TASE 200 is similar to the TASE 150 installed on GTRI's aircraft, but includes an IR camera.

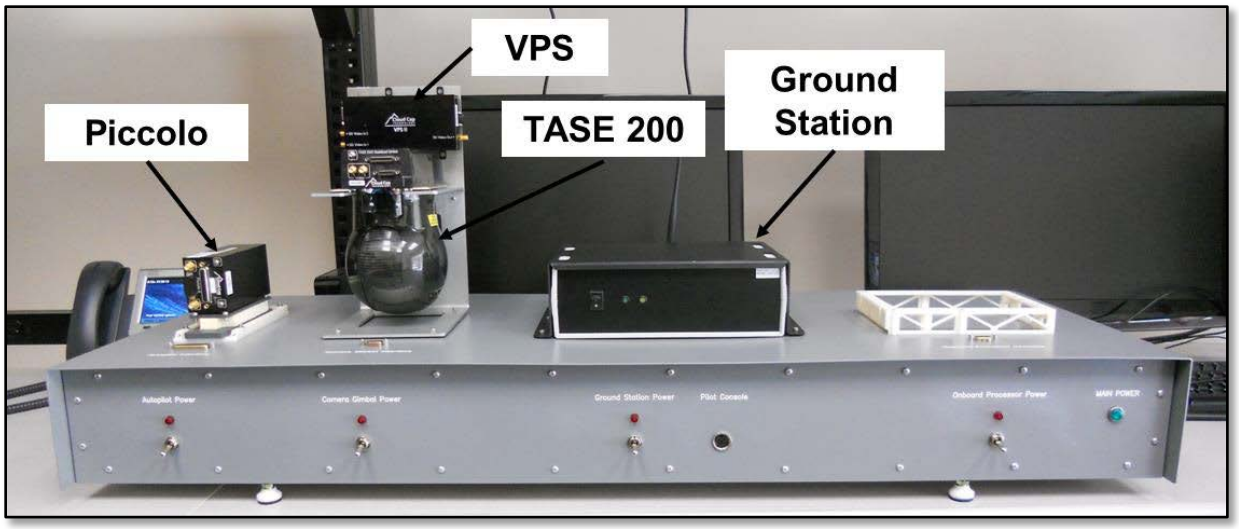


Figure 2. The GTRI HiL emulation environment.

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 may be used to detect the injection of a cyber-attack. To conduct this monitoring GTRI developed a snooping device based on the Raspberry Pi single compute board (SBC) (see section 3.3.1.2). These snoopers can intercept the serial data, decode the information, and retransmit the data. The Raspberry Pi contains two serial ports that allow receiving, altering, and retransmitting of serial data. These monitor points will provide sites where the attack can be injected during the actual flight tests.

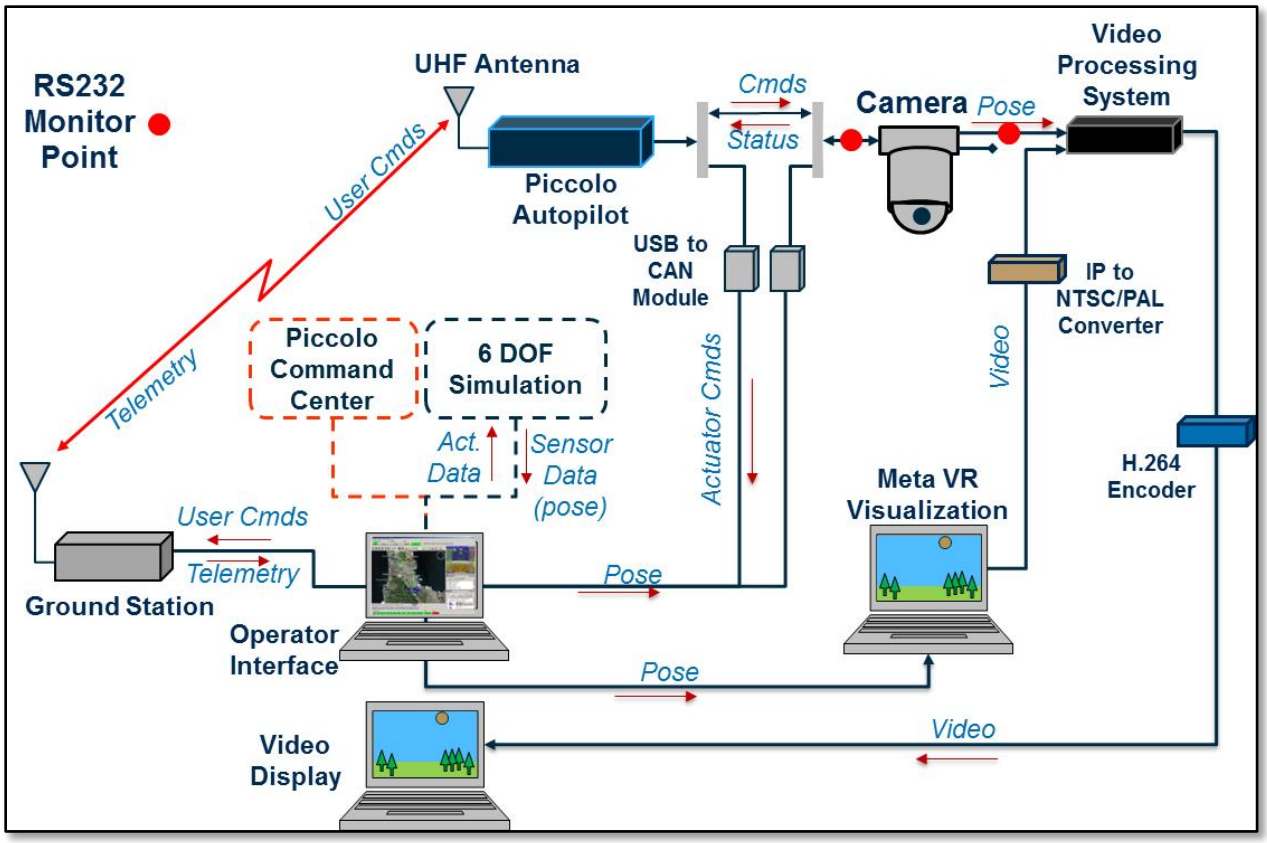


Figure 3. Data flow diagram for the GTRI HiL emulation environment.

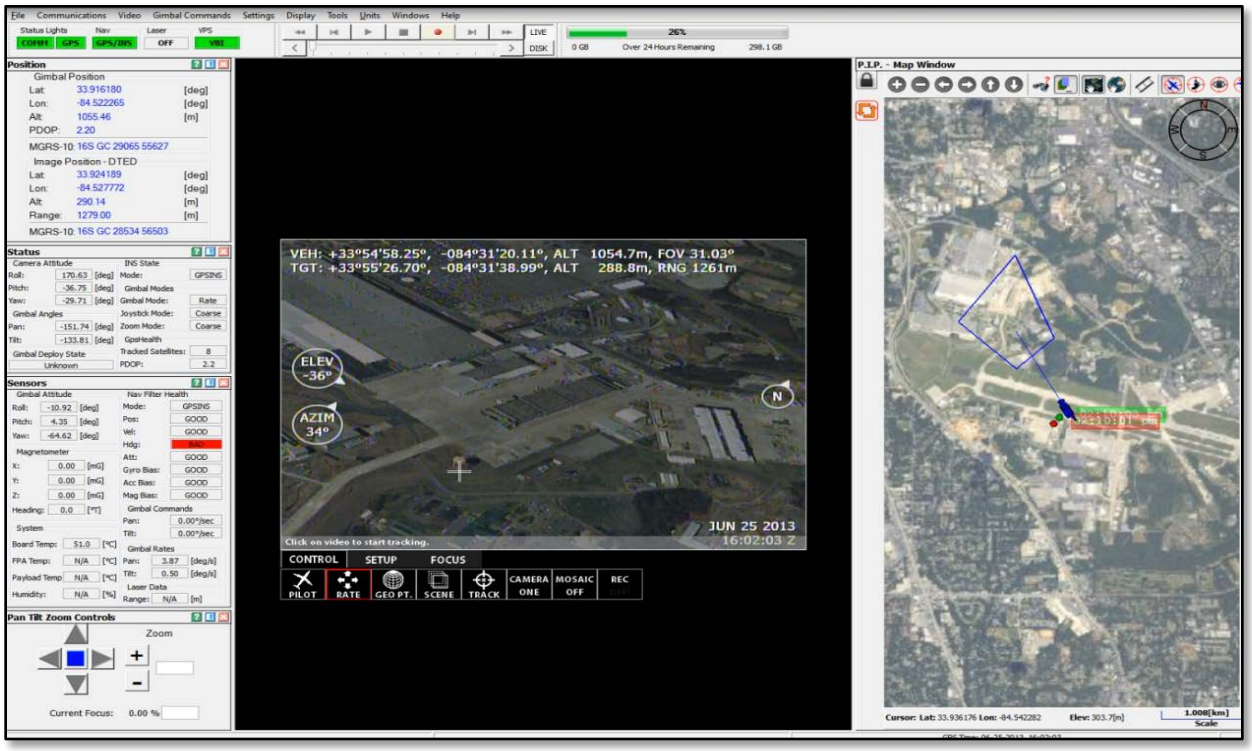


Figure 4. ViewPoint user interface for streaming video created by the MetaVR scene generator

In preparation for an eventual flight test GTRI improved the 6DOF modeling parameters to better represent the GAUSS UAV. This activity included building a higher fidelity inertial model by disassembling the major components of the air vehicle and weighing them. The component weight data was then input to the inertial model spreadsheet provided by Cloud Cap to calculate the aircraft's mass properties (e.g. location and moments of inertia). To improve the aerodynamic modeling the aerodynamic and flight dynamic analysis code AVL from MIT was used to generate the linearized aerodynamic model parameters for the nominal flight state. AVL employs an extended vortex lattice model for the lifting surfaces, together with a slender-body model for fuselages and nacelles. The CAD model of the GAUSS airframe provided by Griffon Aerospace (the manufacturer of the GAUSS airframe) was used for the AVL analysis. The aerodynamic parameters from AVL were input to the Piccolo 6 DOF simulation and the aircraft's dynamic responses to manual control inputs were evaluated by pilots familiar with the handling qualities of the GAUSS aircraft. Several of the lateral and directional stability derivatives were adjusted to make the simulation model match the behavior observed during previous flight tests. To improve the performance modeling, the engine look-up table was updated with results from dynamometer testing of the DA-150 engine used on GAUSS. In addition, the propeller performance model was updated based on performance calculations for the Xoar 28x14 propeller. The propeller performance data was provided by an earlier blade element analysis using the Comprehensive Analytical Model of Rotorcraft Aerodynamics and Dynamics from Johnson Aerospace (CAMRAD/JA). Additional improvements to the simulation model include rebuilding the .xml model file, developing a visualization model of the GAUSS, and updating the simulator autopilot software to v2.2.1.c.

3 Review of Phase I Activities

3.1 Relational System-Aware Cyber Assessment Methodology

The relational System-Aware cyber assessment methodology is a process that has been developed as part of this project to identify the critical system components for a particular system, identify the possible attack paths to attack those components, determine which of those attack paths would be most desirable an adversary, identify possible cyber security defenses against those attacks as well as evaluate the impacts of those defenses on the attacker, 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 assessment methodology is composed of six steps; each step having a well-defined goal, required deliverables, and responsible team(s) for that stage.

This section outlines the relation System-Aware methodology at a high level in order to capture the general flow from one step to another. For more detailed information, refer to Barbara Luckett's 2013 thesis.

3.1.1 Definitions

The relational 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 was 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 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, will analyze costs, to both the attacker and defender, for candidate architectural solutions.

3.1.2 Methodology Process Steps

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

The initial step of the relational 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 evaluation of the solution's collateral impacts is performed by the blue team 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.

3.1.3 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 for this phase 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 Phase 1. We provide an overview of activities for cyber threat analysis efforts in this section which 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.

3.1.3.1 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 5, 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 5. 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 5) 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$_l$). 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) to receive telemetry information about the UAV in flight (the communication system is identified by those components in the green circle in **Figure 5**). 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 5 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.

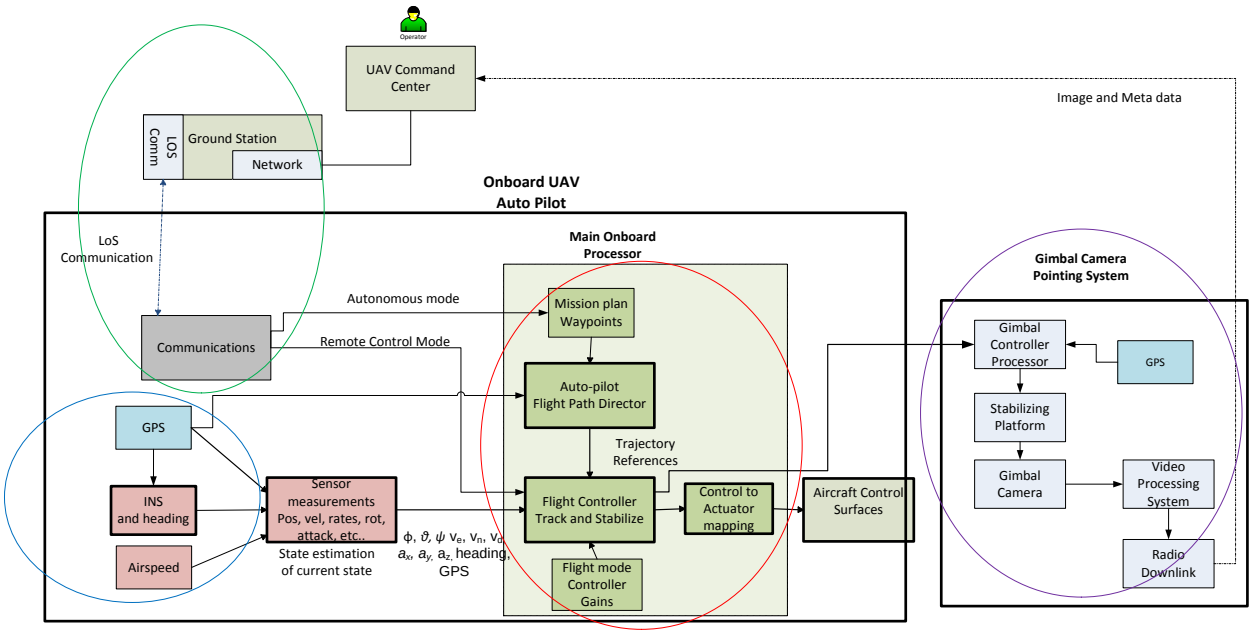


Figure 5. UAV Onboard Systems showing the four major system groups.

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 6, a DAG includes a set of *nodes* and a set of *edges* connecting the nodes. In the system influence diagram shown in Figure 6, 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.

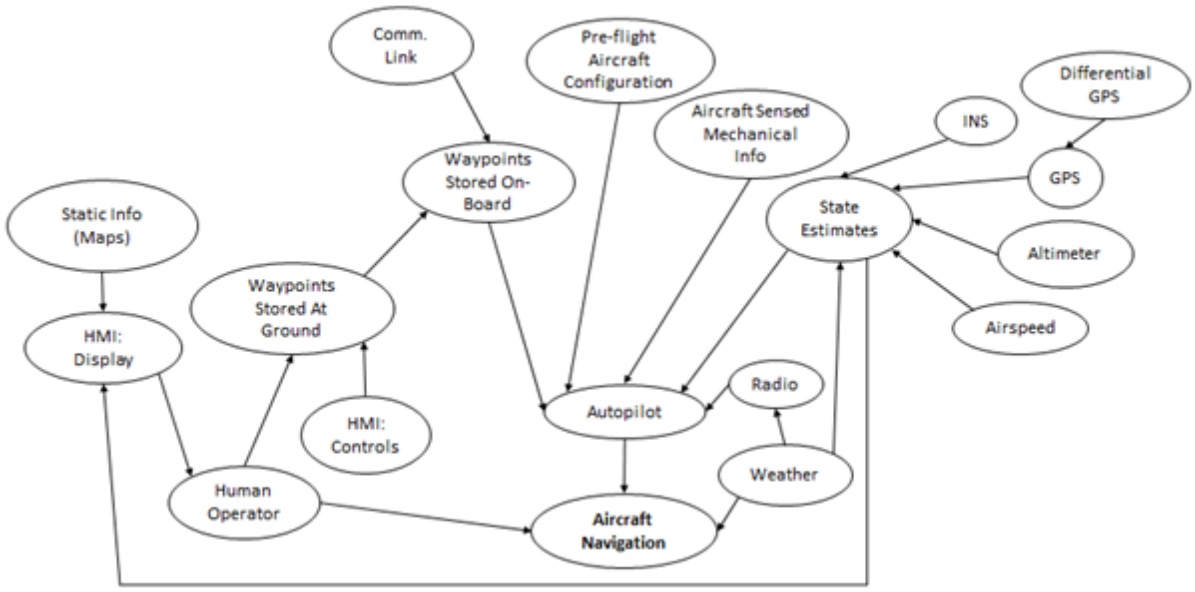


Figure 6. Influence diagram for used to understand the relationship between the UAV subsystems used for navigation.

Figure 6 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 shown on the display influences the actions of the human operator.

Figure 7 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.

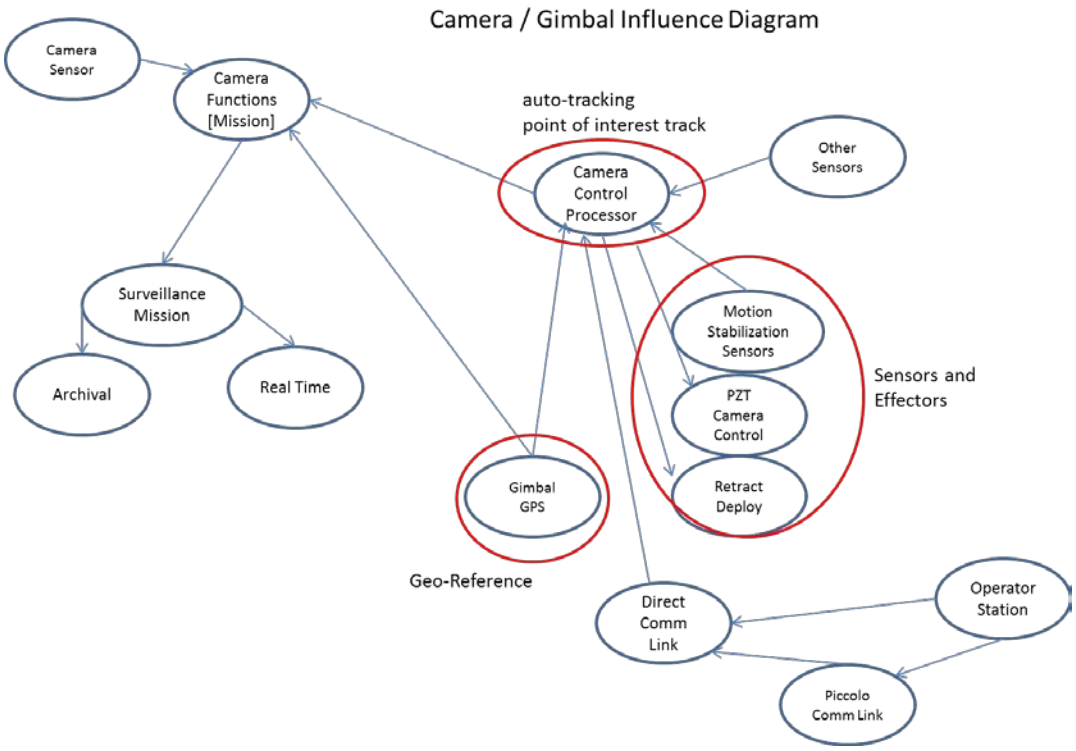


Figure 7. Influence diagram used to understand the relationship between the UAV subsystems used for gathering surveillance data.

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 6 and Figure 7, three major subsystems have been identified for further analysis:

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

Each of these will be discussed in section 3.2.

3.1.3.2 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 8 shows the taxonomy model. Each node at level 2 of the tree (the tree in Figure 8 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 OBJECTIVE by means of PROPAGATION from a lifecycle ORIGIN using malicious ACTIONS exploiting a VULNERABILTY on a RESOURCE/TARGET can change system EFFECTS that have system CONSEQUENCES

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.

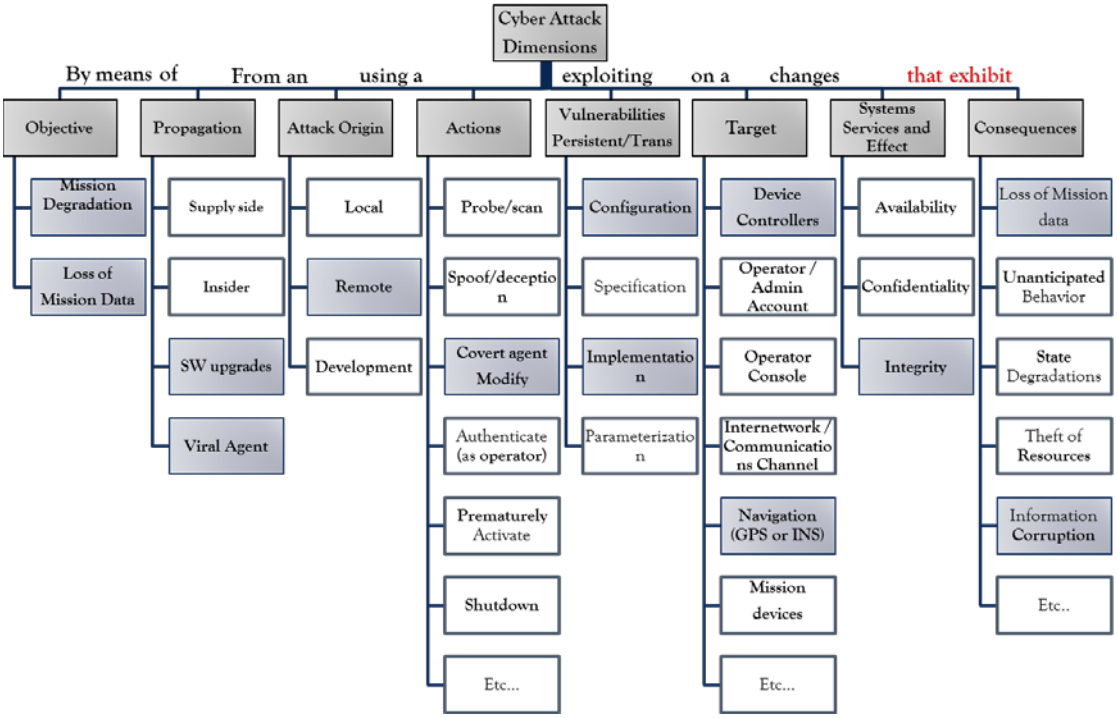


Figure 8. Cyber-taxonomy.

3.1.3.3 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:

- 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.

3.1.3.4 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 9 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.

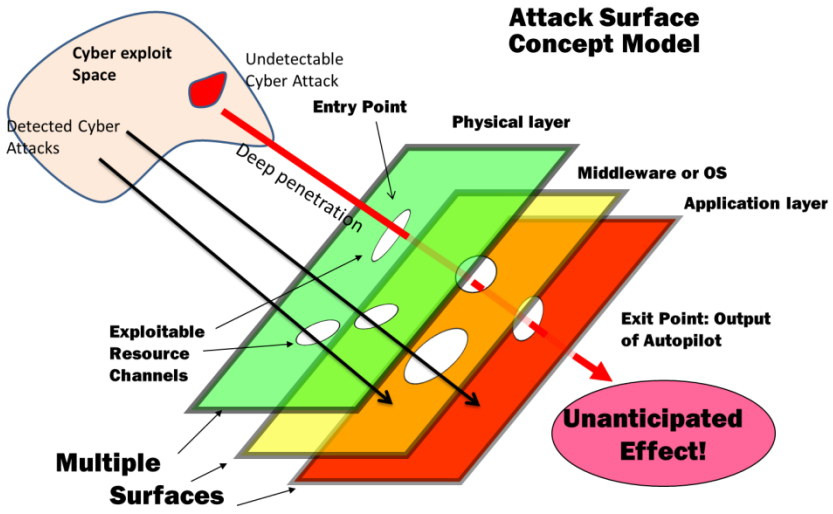


Figure 9. Attack Surface Concept Model.

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.

3.2.1.1 Applying the Relational System-Aware Assessment Methodology to the Parameter-Based Attack Scenario

Step 1

As outlined in section 3.1, the methodology begins by determining which system functions are critical and outlining the variables and their influence relationships. At the highest level, the success of the UAV mission is dependent on the success of three separate functions: (1) the system navigating to the correct location, (2) the sensors on-board working to collect the correct surveillance data, and (3) the platform remaining safe and operational throughout the mission. Of those three functions, the aircraft navigation was selected as critical. This decision was made in part because this was the area the UVa team had been focused on for the majority of the project. Using this to provide the initial scoping for the methodology, a system influence relational diagram was constructed for the *aircraft navigation* function using the DAG notation. This influence diagram is shown in Figure 10.

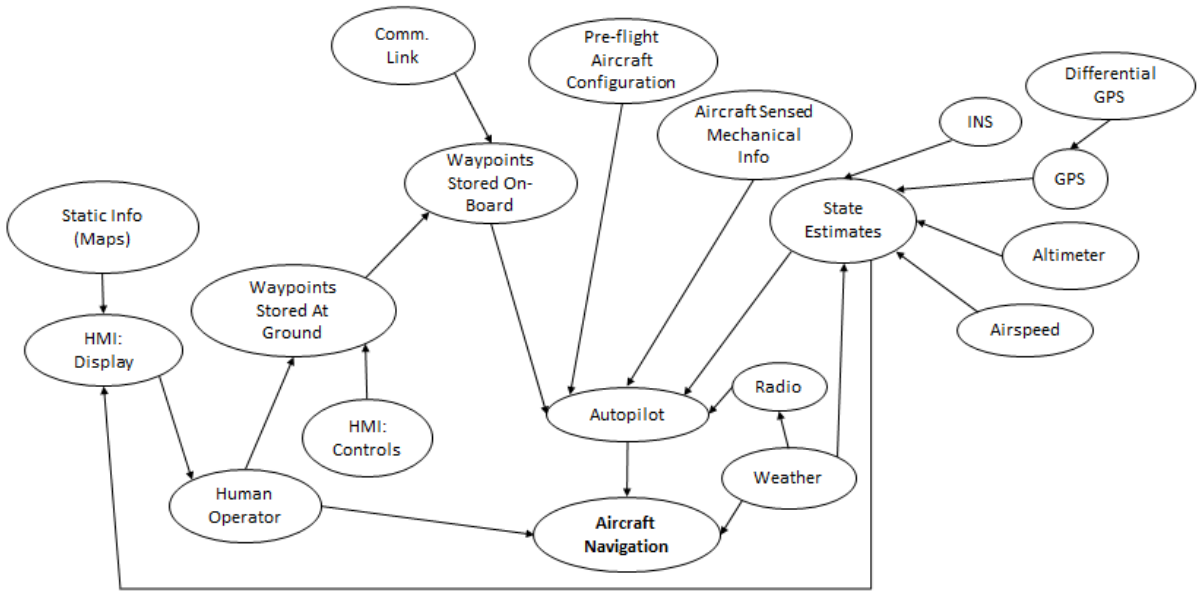


Figure 10. System influence relational diagram for the UAV (aircraft) navigation system.

This graph shows that the outcome of the system function (i.e., the success or failure of the aircraft navigation function) is dependent on three factors: (1) the actions of the human operator, (2) the functionality of the autopilot software, and (3) the weather in the region where the platform will be operating. Similarly, the diagram shows that the status of the operator display is influenced by static information such as maps that are stored in the software and variable information of the state estimates that are collected onboard the platform. In turn, the information shown on the display influences the actions of the human operator.

Step 2

The second step of the relational methodology is focused on the introduction of an intelligent adversary. This perspective is captured by the construction of an attack tree. Based on discussions with the project participants, it was determined that a single attack tree would not necessarily represent the whole picture of how an adversary may wish to exploit a particular system function. Specifically, the adversary's desire and motivation for attempting the attack has a large influence on the manner in which the attack is executed. For instance, for a UAV navigation system attack, two different attack paths may exist that could achieve the same outcome (e.g., change the waypoints onboard the platforms), but one of those paths may have a higher likelihood of being detected while the other has a higher likelihood of success. Thus, an attacker more concerned with remaining undetected may choose the attack path that is harder to detect but offers greater assurances the adversary will not be caught. By dividing the attack structure into multiple trees the team is able to incorporate the adversary's preferences and motivation as well as consider the value vs. detectability trade-off that is often present in the cyber-attack field.

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 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 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.

Figure 11 shows the top level overview of the Concealed Major Trajectory Deviation (i.e., the Concealed Major Trajectory Change) attack tree constructed in step two. Due to size constraints, the lower portions of the subtrees have been rolled up in Figure 11. There are two example subtrees displayed here; the *Alter Waypoints* attack and the *Conceal Flight Trajectory* attack. Similar trees were used to evaluate the other attacks considered; i.e., the *Alter Navigation Sensor Information* attack, the *Alter Autopilot* attack, and the *Alter Actuator Controls* attack.

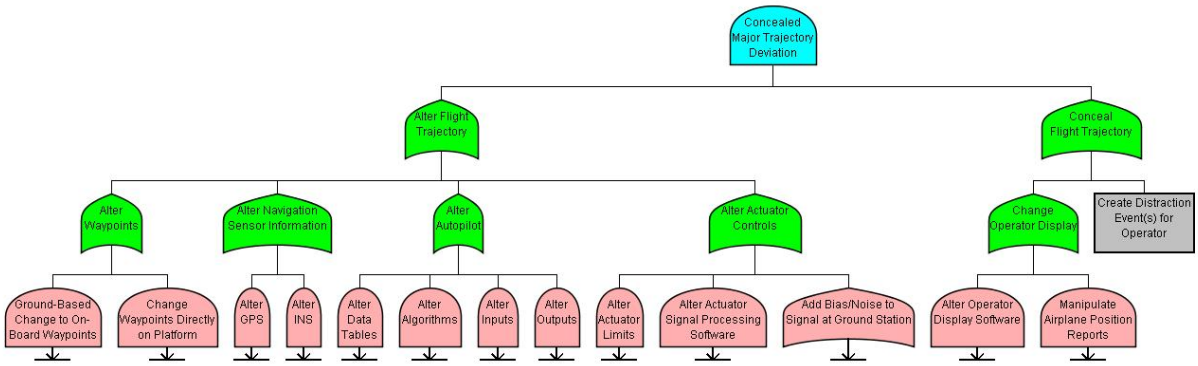


Figure 11. Top-Level overview of *Concealed Major Trajectory Deviation* attack.

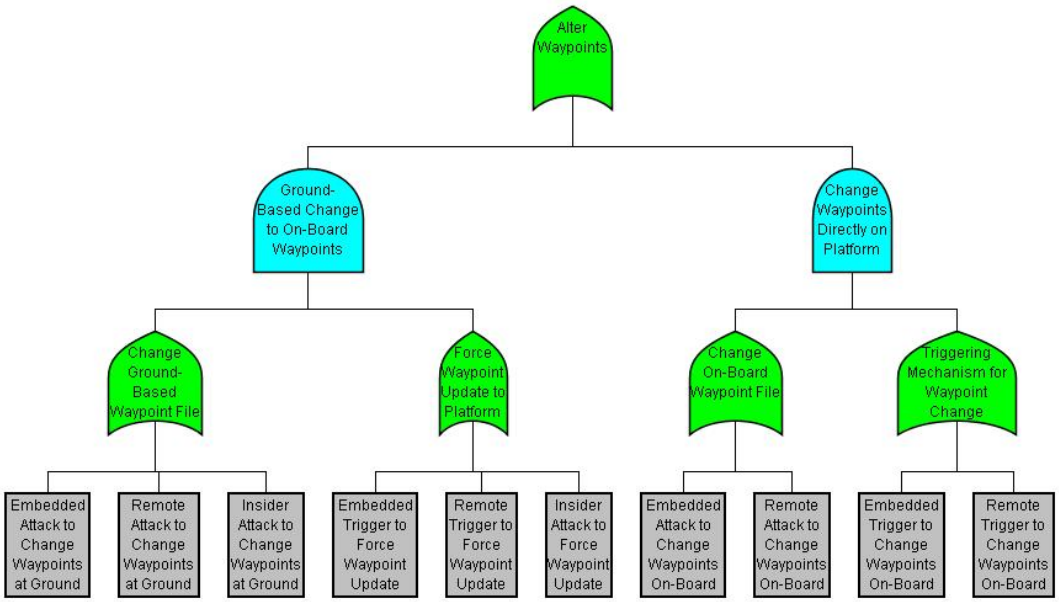


Figure 12. Subtree for *Alter Waypoints* attack.

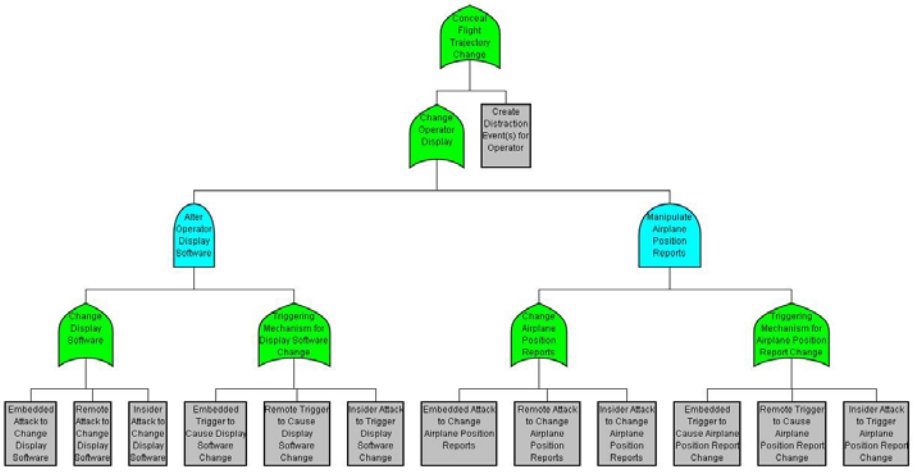


Figure 13. Subtree for *Conceal Flight Trajectory* attack.

As seen by the size of the attack trees represented in the figures above, the attack on the UAV navigation system exist on a large scale. There are a total of 55 leaf nodes in the Concealed Major Trajectory Deviation attack tree alone, which can be executed in various combinations to create a total of 817 possible attack scenarios. Additionally, many of the leaf nodes could be broken down even further in to more specific sub-trees detailing their execution. However, the scope of this tree is more than adequate for the purpose of the relational methodology. It is also impossible to protect against every conceivable attack; thus, we must identify those attack vectors that an adversary could perceive as both relatively simple exploit and of high value.

A couple of trends became apparent during the construction of the Concealed Major Trajectory Deviation attack tree. First, almost every attack strategy included requires two distinct actions: (1) an action to implement the ability to make the desired change and (2) an action to trigger the change on/off as needed. To capture this concept, each attack is represented by an AND node with two subsequent nodes associated to it—one node representing the change action and another represent the triggering mechanism used to initiate the change action. For instance, if the adversary is able to insert a compromised chip into the system at some point in the supply chain they must also have some method to activate the chip; without a triggering mechanism, the infected software is either always on—which could lead to a higher likelihood of the attack being detected and prevented—or is never turned on—which makes the attack worthless. A second trend identified from the construction of the attack tree was the repetition of several different categories of leaf nodes:

1. Embedded: Where infected hardware/software was added to the system at some point in the production process, typically because of a compromised source in the supply chain.

2. Remote: Where the attack is executed from outside of the system, either through an existing link to the system or a completely external factor.

3. Insider: Where an individual who has access to critical aspects of the system and detailed-non-public domain knowledge takes action(s) to exploit the system.

4. Miscellaneous: Attack actions that do not fit any of the three previous designations; for instance, an external decoy (spoofing) or causing a distraction event for the operator.

Step 3

As noted previously, it may be infeasible for the blue team to attempt to protect the system against all 55 attack actions identified in step two. Thus, it is desirable to focus the blue team's efforts on protecting against those attacks perceived as high value to an attacker. The task of identifying a subset of leaf nodes that are perceived as high value to a specific attacker is tasked to the green team. This filtering process involves assessing both the leaf nodes and a potential adversary profile with regards to a set of behavioral indicators, so step three begins by determining the set of necessary behavioral indicator variables to use. Table 1 shows the final set selected for the UAV navigation system attack example.

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?** **<u>Note on possible values:</u>** **Low = entirely unnecessary** **Medium = helpful but not required** **High = impossible without**
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?**

		Note on possible values: **Low = entirely unnecessary** **Medium = helpful but not required** **High = impossible without**
Manpower/Time	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What level of manpower and time is required to successfully complete the attack action?**

Table 1. Behavioral indicator variables names, possible values, and meanings for leaf node assessment.

To assess these values for the set of leaf nodes, a *Behavioral Indicator Variables Assessment* table was created (shown below in Table 2). This table can be reproduced for each of the 55 nodes, and the leaf node name can be inserted in the first line for easy identification. Members of the green team are then tasked with completing the tables.

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**

Table 2. Behavioral Indicator Variables Assessment table.

Several different levels of granularity for the behavioral indicator scales were considered before deciding on a five point Likert-style scale. Using three (Low, Medium, and High) did not allow enough variation while seven (Very Low, Low, Low-Med, Medium, Med-High, High, and Very High) was too many choices and caused the participants to become overwhelmed and revert towards the simplest three (Low, Medium, and High) in many cases.

In addition to making judgments regarding the resources required for the various leaf nodes, step three also includes an assessment of the resources a particular threat actor is expected to possess. Using the same behavioral indicator variables that are used for assessing the nodes (shown again in Table 3 to reiterate their meaning in assessing an adversary), an adversary profile can be constructed.

Behavioral Indicator Name	Possible Values	Meaning for Adversary Assessment
Design Knowledge	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**To what level do you expect the attacker to have access to design knowledge of the system?**
Attack-Specific Technical Ability	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What level of attack- specific technical ability do you expect the adversary to possess?**
Resources	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What level of resources (i.e.: facilities and equipment) do you expect the attacker to have access to?**
Insider Presence (Operational)	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What is the likelihood that the attacker has an insider present in the operational phase of the system?**
Insider Presence (Supply Chain)	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What is the likelihood that the attacker has an insider present at some point in the supply chain?**
Manpower	• **Low** • **Low-Med** • **Medium** • **Med-High** • **High**	**What level of manpower (i.e., time and number of people) do you expect the attacker to possess?**

Table 3. Behavioral indicator variables names, possible values, and meanings for adversary capability assessment.

These profiles are then used to prune the attack tree—a process that eliminates attack scenarios (and thus, leaf nodes) to create a reduced tree that is specific to an individual threat actor. In future applications, a single adversary profile would be created for the threat actor that the group was most concerned with. However, for this example, it was decided that there were four potential classes of adversaries; each adversary possessing vastly different resources, skills, and motivations. Furthermore, all four adversary profiles were constructed in order to highlight the differences among them:

1. Rogue Insider: An individual that has a specific insider connection to the UAV system (i.e., an operator or avionics engineer) and has decided to take action against the

system. He/she has specific knowledge about or access to the system, but their knowledge is more likely to be narrow in scope and they are severely limited in regards to manpower. The individual may not have a particular purpose in attacking the system.

2. Terrorist Group: A motivated, moderately sized group that is working to compromise the system for a reason; e.g., preventing the platform from collecting surveillance information in a specific geographic area to prevent the detection of some activity.

3. Nation State: A country with considerable resources, manpower, and skills that is attacking the system for a reason.

4. Criminal Cyber Group: A moderately sized mercenary group whose goal is to make a profit through the use of one or more attack capabilities. For example selling an attack capability to a terrorist group or nation state to use against the U.S. Criminal cyber groups typically select a target system to which they possess existing access or knowledge.

Table 4 shows the assessments made for each threat actor profile in regards to the six behavioral indicator variables.

	Rogue Insider	Terrorist Group	Nation-State	Criminal Cyber Group
Design Knowledge	**Med-High**	**Medium**	**Med-High**	**High**
Attack-Specific Technical Ability	**Med-High**	**Medium**	**Med-High**	**High**
Resources	**High**	**Medium**	**Med-High**	**Medium**
Insider Presence (Operational)	**High**	**Med-High**	**Medium**	**Medium**
Insider Presence (Supply Chain)	**Low-Med**	**Medium**	**High**	**Medium**
Manpower	**Low**	**Med-High**	**High**	**Med-High**

Table 4. Adversary Profiles Showing Assessed Behavioral Indicator Levels.

Utilizing the evaluations of the adversary behavioral indicators, we were then able to prune the attack trees to a final list of attack scenarios against the UAV navigation system:

- Embedded Attack to Change Waypoints at Ground
- Remote Attack to Change Waypoints at Ground
- Embedded Trigger to Force Waypoint Update
- Remote Trigger to Force Waypoint Update
- Embedded Attack to Alter INS Measurements
- Embedded Trigger to Cause Alteration of INS Measurements
- Embedded Attack to Change Parameter Data Tables
- Remote Attack to Change Parameter Data Tables
- Embedded Trigger to Cause Parameter Data Tables

- Remote Trigger to Cause Parameter Data Table Change
- Embedded Attack to Add Bias to I/O Commands at the Ground Station
- Remote Attack to Add Bias to I/O Commands at the Ground Station
- Embedded Attack to Change HMI Display Software
- Remote Attack to Change HMI Display Software
- Embedded Trigger to Cause Display Software Change
- Remote Trigger to Cause Display Software Change
- Remote Attack to Change Airplane Position Reports
- Embedded Trigger to Cause Airplane Position Report Change
- Remote Trigger to Cause Airplane Position Report Change

Step 4

This reduced set of 19 nodes can be compared with the existing portfolio of design patterns to determine which design patterns are the most applicable. A design pattern can be considered applicable for several reasons. It can make a leaf node more difficult, uncertain, or expensive, or make it so the completing the action requires that the adversary have specialized skills or equipment. Additionally, a design pattern may be applicable if it increases the likelihood that the attack can be detected and prevented. Finally, a design pattern may be applied if it decreases the consequences on the defense team given the attack is still successful.

The 19 remaining leaf nodes can be grouped into six categories: (1) ground station based waypoint change, (2) change of INS measurements, (3) change of parameter data tables, (4) addition of bias/noise through the I/O commands at the ground station, (5) spoofing the human machine interface (HMI) operator display through a software change, and (6) manipulation of the airplane position report. Table 5 shows several possible design patterns that were determined to be applicable for each of the remaining six attack types.

Attack Type	Design Pattern	Detailed Description of DP Functionality
1. Ground Station Based Waypoint Change	**Parameter Assurance**	**Typically, there will be a pre-loaded flight plan based on the mission. Parameter Assurance compares the waypoints input at the ground to the table of values in the system to check for large, unexpected deviations.**
	Data Consistency Checking	**A change of the waypoints at the ground station needs to follow a step order of steps. Data Consistency Checking looks to see where the change originated from to verify that it was initiated by the operator.**
2. Change of INS Measurements	**Diverse Redundancy**	**Diverse Redundancy involves the implementation of additional INS devices, from diverse manufacturers/suppliers.**
	Physical Configuration Hopping	**Physical Configuration Hopping involves**

		hopping between multiple INS components at a pre-determined interval.
	Verifiable Voting	**Voting involves comparing the values returned by multiple INS devices to identify and isolate a compromised INS.**
3. Change of Parameter Data Tables	**Data Consistency Checking**	**A change of the parameter data tables needs to follow a step order of steps. Data Consistency Checking looks to see where the change originated from to verify that it came from a trusted source.**
	Parameter Assurance	**Parameter Assurance compares the parameter data table values to a table of preexisting "gold standard" of flight control values in the system to check for large, unexpected deviations.**
4. Addition of Bias/Noise Through I/O Commands at Ground	**State Estimation**	**State Estimation uses existing mechanisms in the system to estimate other state variables in the system. These values can be used indirectly to infer what the state in question should be.**
5. Spoofing the HMI Operator Display Through Software Change	**Data Consistency Checking**	**A change of the HMI display software needs to follow a step order of steps. Data Consistency Checking looks to see where the change originated from to verify that it came from a trusted source.**
	Parameter Assurance	**Parameter Assurance involves using a back-up system (possibly considerably more rudimentary than the main operator display) to collect the same information as the main display. These values may not be displayed to the operator, but the system compares the main display values to those collected by the back-up display system to check for deviations.**
	State Estimation	**State Estimation uses existing mechanisms in the system to estimate other state variables in the system. These values can be used indirectly to infer what the operator display should be showing.**
6. Change of Airplane Position Report	**Diverse Redundancy**	**Diverse Redundancy involves the implementation of additional radio devices, from diverse manufacturers/suppliers (the radio is used as an example here because it is the source that sends the position information from the platform to the ground**

		station, but diverse redundancy could be added to another device earlier in the process to accomplish the same outcome).
	Physical Configuration Hopping	**Physical Configuration Hopping involves hopping between multiple radio components at a pre-determined interval (see note regarding diverse redundancy above: the radio is only an example for one device where physical configuration hopping could be implemented).**
	State Estimation	**State Estimation uses existing mechanisms in the system to estimate other state variables in the system. These values can be used indirectly to infer what the state in question should be. State Estimation would only work in this situation if the change caused by the adversary did not affect all of the state estimates included in the Airline Position Report (the design pattern relies on having some secure estimates to use in order to infer less secure estimates).**

Table 5 Applicable design patterns for each attack type.

At this point, these design patterns can be inserted back into the system relational influence diagram constructed in the first step to understand how the different defensive strategies complicate the actions required by the adversary and how they interact to provide multidimensional coverage of the system. Several examples are discussed here to demonstrate the type of insight that can be gained in this step.

The first defensive strategy to consider is the addition of *Parameter Assurance* implemented on the waypoints stored at the ground station. This design pattern works by maintaining access to a pre-loaded flight plan associated with the mission and comparing the waypoints at the ground to these stored values to check for large, unexpected deviations from the expected waypoints. In the initial minimal defense system depicted in the influence relational diagram from step one, if the attacker wanted to execute a ground-based attack of the waypoints, they had to conduct an attack to change the values at the ground and also create a trigger to force the waypoints to update to the platform (both of which could be embedded within the system through supply chain infiltration or done remotely.) After the hypothetical implementation of this design pattern, the attacker still has to do both of those actions, but they now also need to consider two additional elements. Adding Parameter Assurance inserts two new nodes into the system influence relational diagram: the *Preloaded Flight Plan Values* and the *Parameter Deviation Checking Mechanism*. In order to successfully execute the attack with Parameter Assurance implemented, the adversary still needs to alter the ground station waypoints but also now needs to do one of the following:

1. Change the preloaded table of expected waypoint values associated with the flight plan to match their manipulated waypoints values so that the functioning comparison mechanism will return that the values are the same.

2. Alter the parameter deviation checking mechanism so that it will not report that there is a large deviation between the two sets of waypoints.

This increases the complexity for the adversary to successfully execute the attack. As we continue to consider the additions of other protections, including diverse redundant components, configuration hopping and verifiable voting, we continue to alter the asymmetry of the attack vector making it increasingly difficult for an adversary to be able to affect change on all of the components within the system that would be necessary to execute the attack.

Step 5

Step five is focused on evaluating the potential design patterns from step four in regards to their impact on the defensive team. This impact can be categorized into three criteria: (1) implementation cost, (2) lifecycle cost, and (3) collateral system impacts. Similar to the behavioral indicator variables assessed in step three, a five point Likert scale (with possible values of Low, Low-Med, Medium, Med-High, and High), along with an optional notes section for justifying comments, was used here for ease of evaluation. Table 6 shows the levels assessed for each design pattern regarding implementation cost, lifecycle cost, and collateral system impacts.

Attack Type	Design Pattern	Implementation Cost	Lifecycle Costs	Collateral System Impacts
Ground Station Based Waypoint Change	**Data Consistency Checking**	**Medium**	**Low-Med**	**Low**
	Parameter Assurance	**Low-Med**	**Medium**	**Medium**
Change of INS Measurements	**Diverse Redundancy and Physical Configuration Hopping**	**Medium**	**Medium**	**Medium**
	Diverse Redundancy and Verifiable Voting	**Low-Med**	**Medium**	**Low**
Change of Parameter Data Tables	**Data Consistency Checking**	**Low-Med**	**Med-High**	**Low**
	Parameter Assurance	**Med-High**	**Medium**	**Med-High**
Addition of Bias/Noise Through I/O Commands at Ground	**State Estimation**	**Medium**	**Low-Med**	**Low**
Change of the HMI Display Software	**Data Consistency Checking**	**Medium**	**Low-Med**	**Medium**
	Parameter Assurance	**Med-High**	**Med-High**	**Med-High**
	State Estimation	**Medium**	**Low-Med**	**Low-Med**
Change of Airplane Position Report	**Diverse Redundancy and Physical Configuration Hopping**	**Med-High**	**Medium**	**Med-High**
	State Estimation	**Medium**	**Low-Med**	**Medium**

Table 6. Implementation Cost, Lifecycle Costs, and Collateral System Impacts associated with each applicable design pattern.

At this point, the set of possible defensive solutions can be reduced one more time based on the budget available for implementing System-Aware security solutions and preferences regarding collateral system impacts. Any design patterns that exceed the budget or have unacceptable collateral system impacts can be eliminated during this step. For instance, if Med-High costs over the lifecycle of the system were deemed to be over budget and Med-High collateral system impacts were deemed to be unacceptable, four alternatives can be eliminated. This would leave eight available alternatives:

1. Data Consistency Checking implemented on the ground station waypoint file to prevent a ground-based waypoint change.

2. Parameter Assurance implemented on the ground station waypoint file to prevent a ground-based waypoint change.

3. Diverse Redundancy and Physical Configuration Hopping implemented on the INS to prevent the alteration of the INS measurements.

4. Diverse Redundancy and Verifiable Voting implemented on the INS to prevent the alteration of the INS measurements.

5. State Estimation implemented to prevent the addition of bias/noise through I/O commands at the ground station.

6. Data Consistency Checking implemented on the HMI operator display at the ground station to prevent a change of the HMI display software.

7. State Estimation implemented on the HMI operator display at the ground station to prevent a change of the HMI display software.

8. State Estimation implemented to prevent the manipulation of the airplane position report.

While any strategies eliminated at this point do not need to be fully discussed in step six, it is important to note that they have not been completely disregarded. Design patterns eliminated during this step will be documented so they can be revisited at a later point if the budget or the team's views on certain collateral system impacts changes.

Step 6

Going into the final step, there are eight possible defensive strategies to consider, all of which increase the difficulty for a specific adversary to complete one of their most preferred attack actions while remaining at an acceptable impact to the defense. While eight is much more reasonable than the entire original set of possibilities, it is still more than what the design team can afford to implement. Step six involves all of the project participants coming together for a collaborative discussion focused on weighing the security trade-offs that exist among these remaining alternatives in order to select a subset to be implemented as part of the final solution. There are four factors that should be considered when piecing together the final cohesive security architecture: (1) budget, (2) coverage, (3) multi-dimensionality, and (4) asymmetry.

Budget is used as the initial prescreening criteria here. The project team as a whole decides that they would like to implement about a quarter of the remaining design patterns and are comfortable with the implementation costs associated with two or three Low-Med or Medium valued solutions. The idea of system coverage is used as a second prescreening filter. After completing all of the analyses throughout the framework, team members will probably have certain alternatives of interest in the remaining set because of the areas of the system they protect or the types of attacks they protect against. For this example, it was determined that the team was more concerned with the flight trajectory change itself than the concealment of that change. Narrowing the coverage scope in this regard made sense because

protecting against the change itself protects the system against all three valued attack trees (rather than just the Concealed Major Trajectory Deviation tree), and if the action to alter the flight trajectory is difficult enough that the attacker can't complete it, the actions to conceal it become unnecessary. Specifically, the team wanted to focus on the possibility that the adversary would alter the navigation via a ground-based waypoint change or alteration of the INS values on board the platform and thus wanted to further examine the options for protecting against these threats. Three possible solutions were constructed to be compared in a more detailed analysis:

1. Implementation of both parameter assurance and data consistency checking on the ground station waypoint file to prevent a ground-based waypoint change.

2. Implementation of both diverse redundancy, physical configuration hopping, and verifiable voting on the INS to prevent the alteration of the INS measurements.

3. Implementation of one defensive strategy on the ground waypoint station and one on the INS device (specifics determined by the analysis of architectures 1 and 2).

These three architectural solutions were analyzed in regards to each one's impact on the adversary relative to its impact on the defense. Since there were six behavioral indicator variables used to assess the attack actions and construct the adversary profiles, the impact of a defensive strategy can be assessed across six different dimensions. Different defensive actions will increase the difficulty for an attacker in different areas, and an optimal solution can be constructed by combining strategies that complement each other in regards to the multidimensionality. The impacts associated with the Parameter Assurance and Data Consistency Checking for Parameter Assurance and Data Consistency Checking Implemented on the Ground Station Waypoint File are listed in Table 7.

Parameter Assurance		**Data Consistency Checking**	
Impact to Defense	**Impact to Adversary**	**Impact to Defense**	**Impact to Adversary**
Implementation Cost: Low-Med	**Design Knowledge: Medium**	**Implementation Cost: Medium**	**Design Knowledge: Medium**
Lifecycle Cost: Medium	**Attack-Specific Technical Ability: Low-Med**	**Lifecycle Cost: Low-Med**	**Attack-Specific Technical Ability: Med-High**
Collateral System Impact: Medium	**Resources: Low-Med**	**Collateral System Impact: Low**	**Resources: Med**
	Insider Presence (Operational): Low		**Insider Presence (Operational): Medium**
	Insider Presence (Supply Chain): Medium		**Insider Presence (Supply Chain): Med-High**
	Manpower/Time: Low-Med		**Manpower/Time: Medium**

Table 7 Impact to Adversary Relative to Impact to Defense for Parameter Assurance and Data Consistency Checking

For demonstration purposes in this project, implementation of the parameter assurance for the waypoint change was chosen by the project team for this application and it was also enhanced to include protection on both the ground and in the air.

3.2.1.2 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, SBC, etc.

For the Phase 1 implementation of the parameter-based attack, the Trojan horse has been implemented on a laptop running Microsoft Windows 7© connected to the Piccolo autopilot over one of the available serial connections. In addition, the Trojan horse is able to be triggered by either entering a specific geographic region, or, to facilitate experimentation, the laptop allows for a malicious user to redirect the aircraft to any waypoint through a simple text based interface. All testing of the attack will be performed using the Piccolo autopilot's HiL emulation capabilities outlined in section 2. The integration of the attack platform into the HiL can be seen in Figure 14.

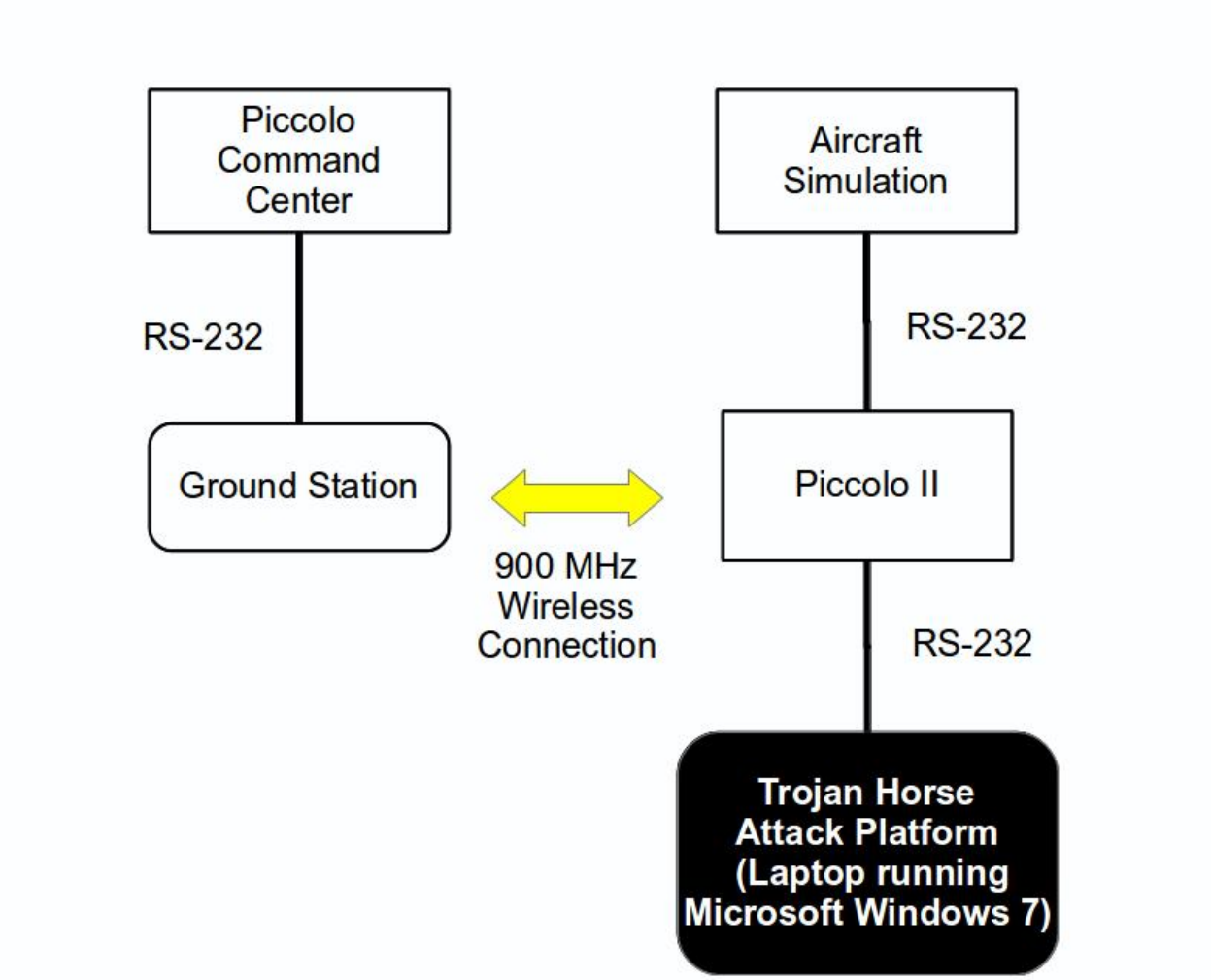


Figure 14. Piccolo II HiL configuration with an Embedded Attack Platform to dynamically direct the UAV to a specified waypoint during flight. The Embedded Attack Platform is a laptop connected to the Piccolo II autopilot via a serial connection using the RS-232 protocol.

3.2.1.3 Masking the Parameter-Based Attack to Prevent User Detection

The attack implementation outlined in section 3.2.1.2 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.

Figure 15 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.

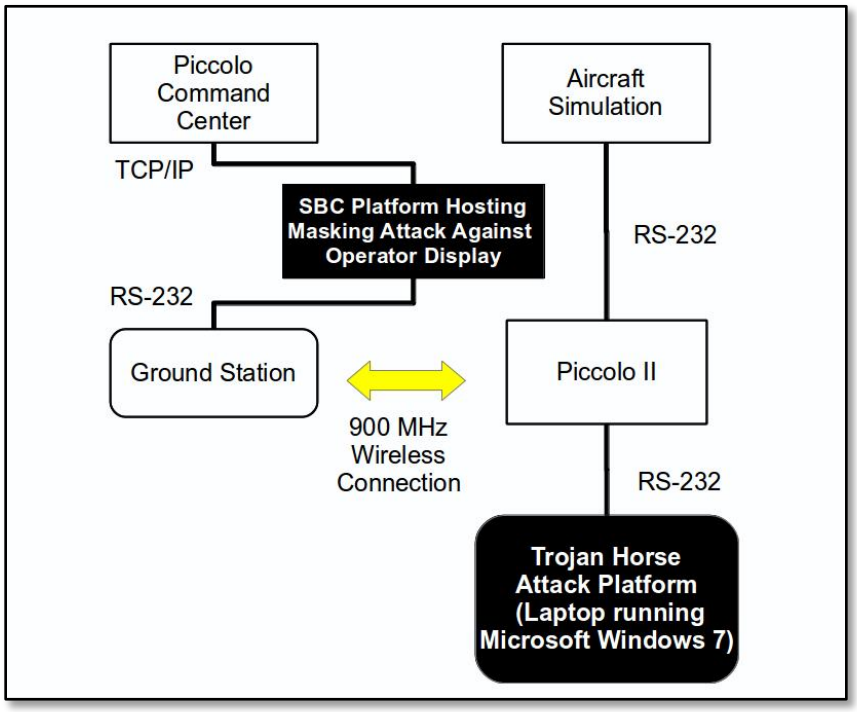


Figure 15. Embedded attack platform for masking the operator display on a SBC.

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

As described in section 3.2.1.3, 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 16 shows the configuration for the HiL emulation environment that initiates a ground-based version of the same parameter-based attack as described section 3.2.1.3. 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.

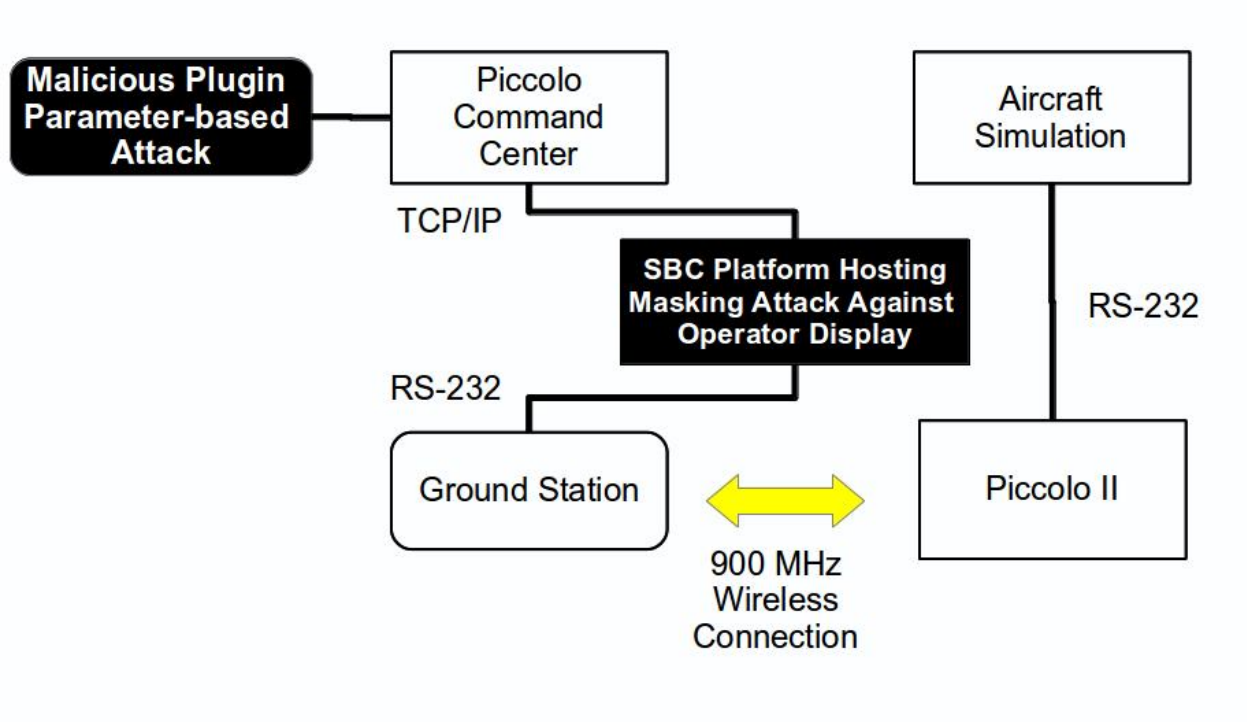


Figure 16. Piccolo II HiL configuration with Embedded Attack Platform to dynamically direct the UAV to a specified waypoint during flight while masking the change from operator display. Attack is a plugin embedded into the PCC. Embedded attack platform for masking the operator display is a SBC that sits between the connection from the PCC and the ground station.

3.2.2 GPS System Attacks

3.2.2.1 Applying the Relational System Aware Assessment Methodology

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 17 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 georeference 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 georeferencing of the image data from the camera.

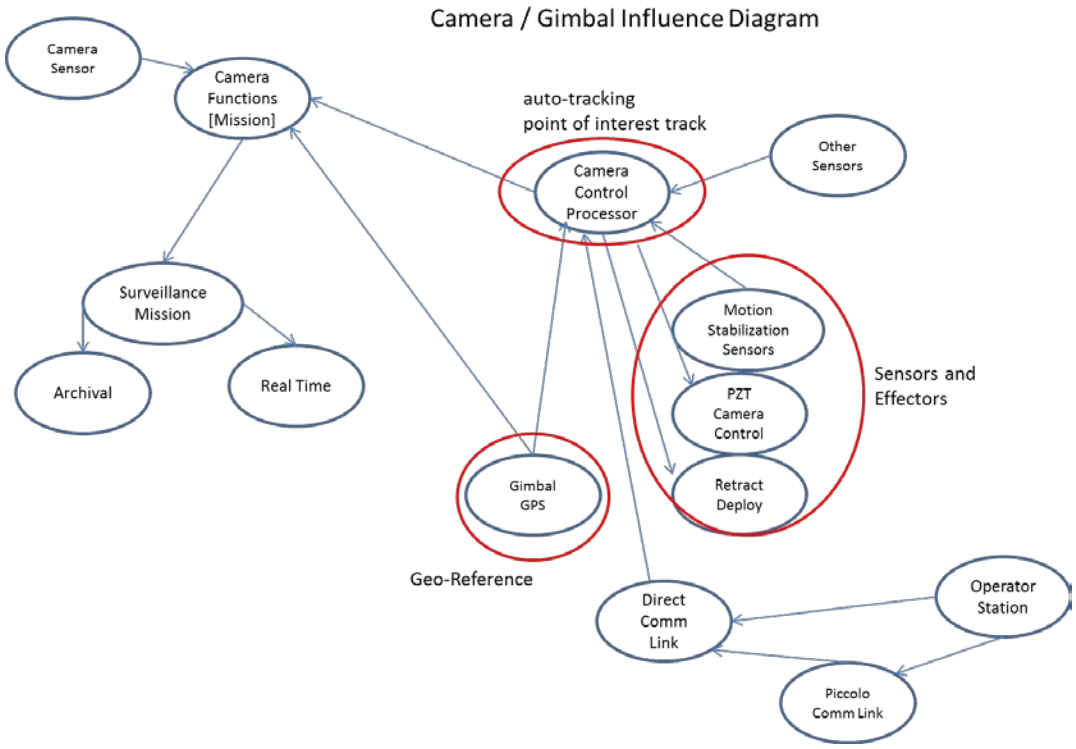


Figure 17. Camera gimbal influence diagram.

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 georeferencing 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 18 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.

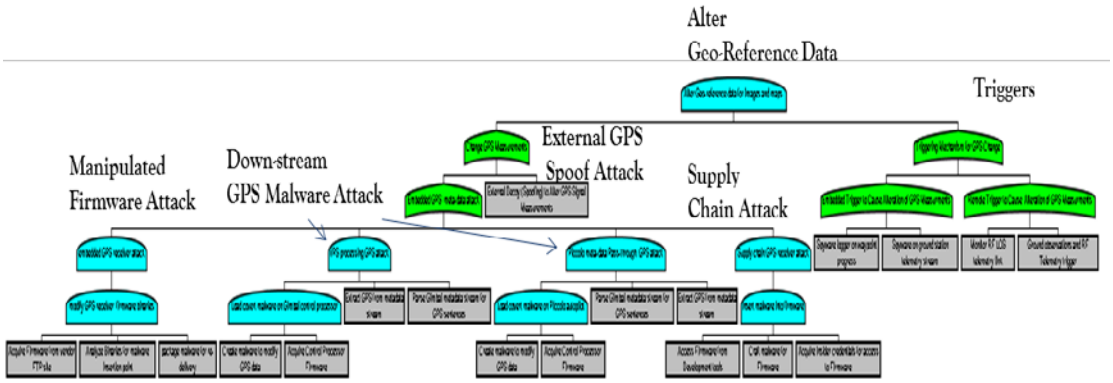


Figure 18. Attack tree for GPS attacks to alter georeference data.

In Figure 18 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 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 data base 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 19, 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.

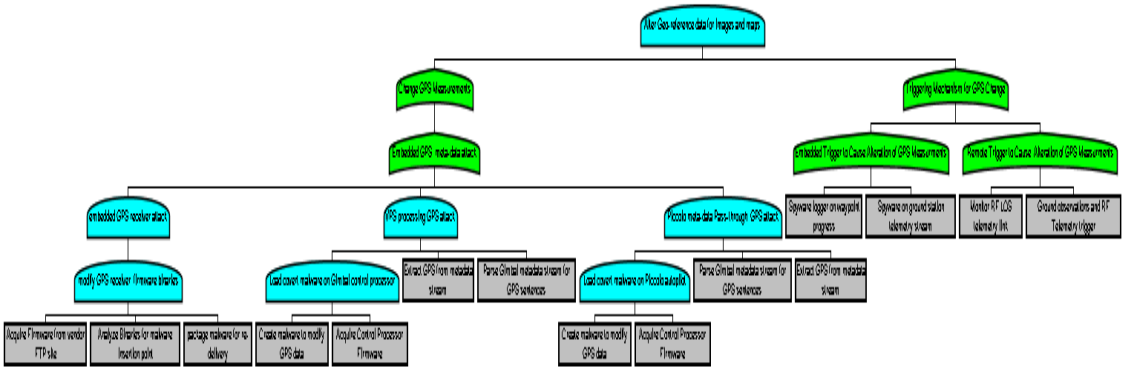


Figure 19. Pruned Attack Tree.

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 8, 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 received measurements 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 8. 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 9. 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 the 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 9. 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 2.

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.

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 20.

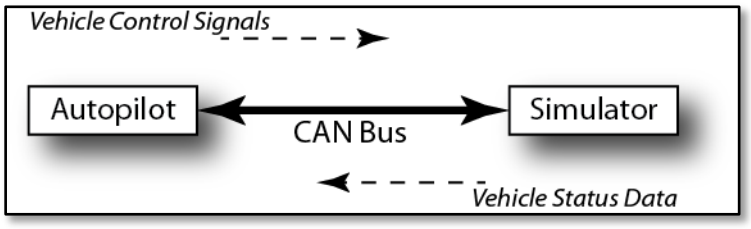


Figure 20. Autopilot to Simulator communications.

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 21.

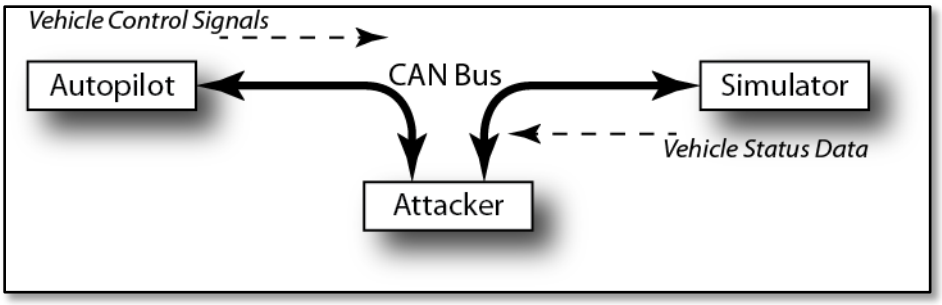


Figure 21. Simulated attack configuration.

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 22 which shows the deviation in the UAV's ground track.

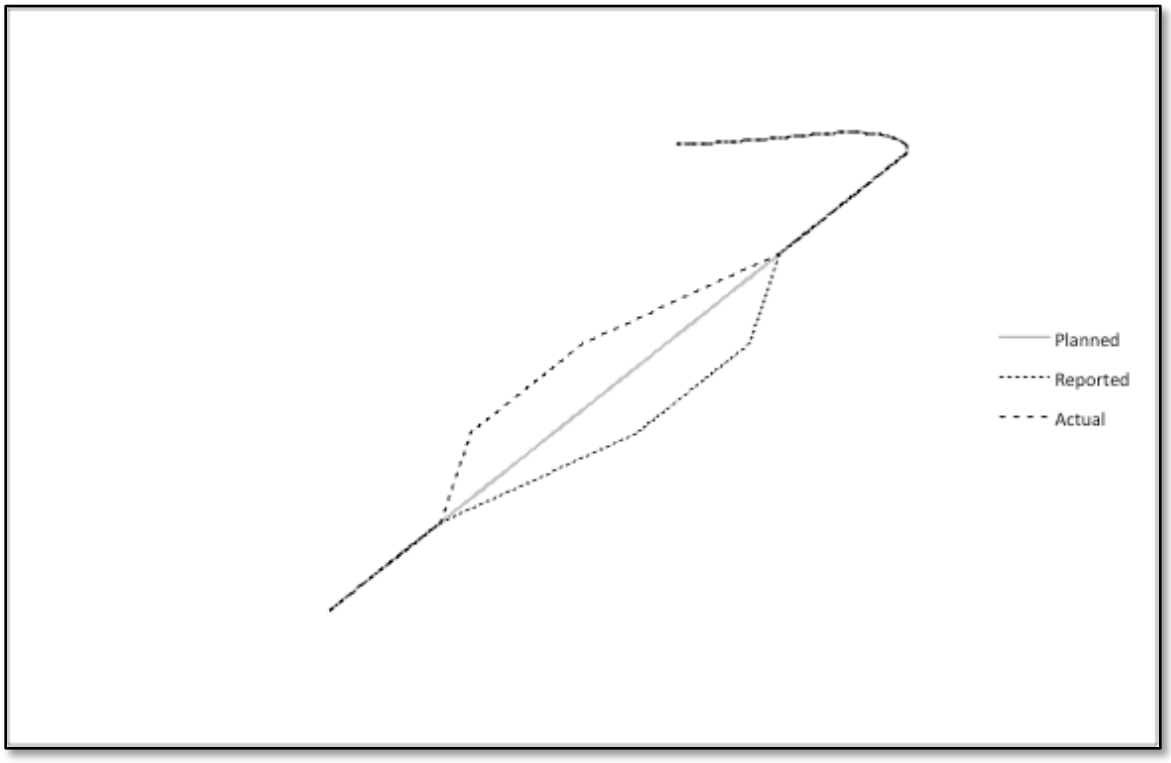


Figure 22. Example of GPS attack.

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

3.2.3 Gimbal System 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 Technology. The following sections describe potential attack vectors for the camera gimbal. Section 3.3.4 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.

3.2.3.3.1 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.

3.2.3.3.2 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.

3.2.3.3.3 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.

3.2.3.3.4 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.

3.2.3.3.5 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.

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 High Level Attack Scenarios

As outlined in sections 3.3 and 4, the proposed prototype Sentinel will need to convert 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 sections 3.3 and 4, 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 reconfigurability and flexibility that makes it favorable for the purpose of prototyping and concept-proving. In this proposal, the soft-cores for protocol conversion and all hardware-based protections will be 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 attack will be a kill switch embedded into the soft-core running on the FPGA. The attack will be triggered by a specific pattern embedded into the RS-232 data stream. The pattern must be long enough to avoid coinciding with normal data content. The triggering pattern will be sent from one of the RS-232 data sources, including the autopilot and camera gimbal.

Figure 23 shows the transmission of an 8-bit RS-232 data frame. The data being transmitted is the ASCII code for the letter K (0x4B), LSB first (Least Significant Bit). The frame starts with a start bit, a logical 0 (represented by a high voltage in Figure 23), and ends with a stop bit, a logical 1 (represented by a low voltage).

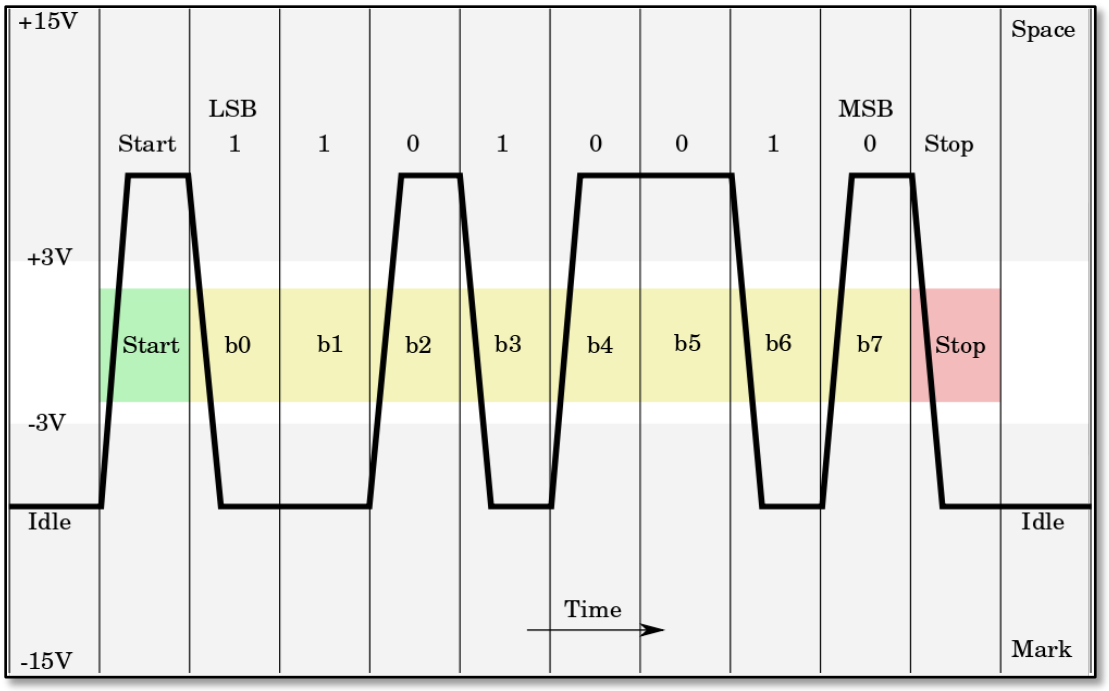


Figure 23. RS-232 Data Transmission.

When the triggering pattern is recognized, a compromised Universal Asynchronous Receiver/Transmitter (UART) interface will perform a kill switch attack by discarding all received data. This can be implemented by small changes in the UART receiver's logic. A more skilled and knowledgeable attacker may be able to insert his own data into the stream and accomplish more sophisticated attacks.

3.3 Design and Development of the Super Secure, Smart 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 CloudShield

The prototype system employs an off-the-shelf network security product called the CloudShield CS-2000 content processing platform (here after referred to as CloudShield) (CloudShield is shown in Figure 24) as the Sentinel Security Platform. While CloudShield includes many of the desirable features of a programmable Sentinel, it does not meet the size, weight, and power requirements for airborne use. However, as seen in the analysis shown in Figure 25, it is a good environment for testing System-Aware security design patterns due to its capacity to perform deep packet inspections of data flowing in and out of the system, with negligible latency issues associated with the inspection process. The CloudShield also provides a platform which allows for redundancy within the network architecture itself. We used the CloudShield environment to develop the security design patterns supporting a ground-based prototype version (based on SiL simulation environment of the Piccolo system) and air-based prototype version (based HiL emulation environment) of the Sentinel which will be re-configured for actual flight in the next phase of this work. This permits the design team to better separate the design topics of cyber security effectiveness and the footprint requirements for flight by first developing effective algorithms and then converting the software to operate on new, flight-capable hardware. CloudShield supported the prototype development approach for the system Parameter Assurance and Data Consistency design patterns.

Figure 24. CloudShield CS-2000 content processing platform with two deep packet inspection modules.

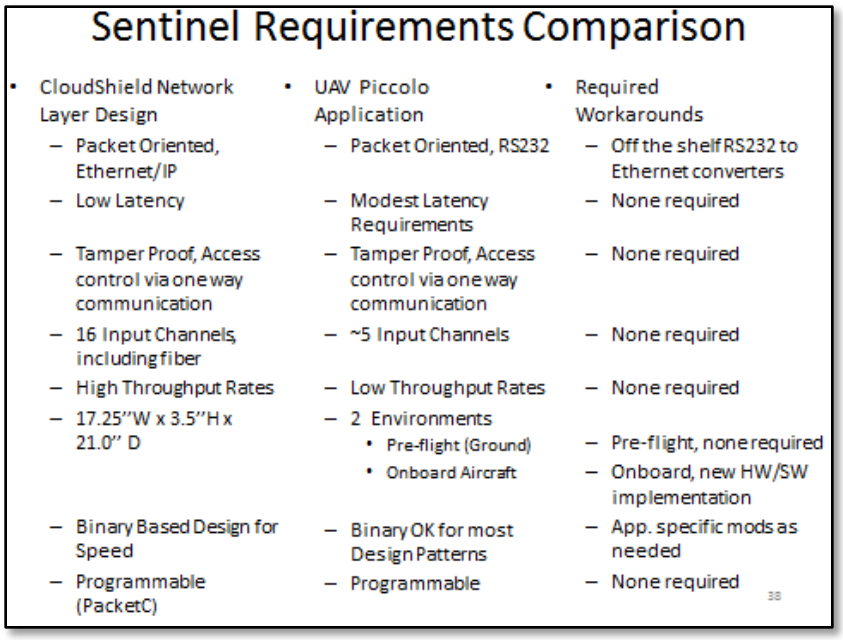


Figure 25. Comparison of CloudShield features to Sentinel requirements.

In order to protect the Sentinel from becoming a potential target for cyber-attacks, the Sentinel design includes security features for the Sentinel itself. For the Phase 1 Sentinel these include the ability to perform HW/SW configuration hopping by leveraging the fact that the CloudShield includes a redundant processing module.

To integrate the CloudShield into the HiL emulation environment, it was necessary to convert information from RS-232 into TCP/IP packets for processing. As seen in Figure 29, this was accomplished through the usage of SBCs (Single Board Computers)--specifically the Phidgets SBC2 and Raspberry Pi. In addition, these SBCs were used to connect the CloudShield to the secure back channel; an 802.11 network in the initial prototype. Finally, the CloudShield system also provided the capability to monitor traffic between the PCC and the ground station. This enabled the initial prototype system to be able to monitor the integrity of the PCC, which, in turn, enabled for the classification of cyber-attacks that altered the UAV flight plan--i.e., directed the UAV to another waypoint--that originated from the ground from those that originated from the autopilot from those initiated by an operator/pilot. With this capability, an additional cyber-attack was created on the PCC. This cyber-attack performs the same functionality as the embedded attack; i.e., it would direct the UAV to fly to another waypoint. However, this attack would originate from a plugin maliciously installed on the PCC. Similar to the embedded Trojan horse, this plugin directs the aircraft to a different waypoint when the UAV enters a specific geographic region(s), as well as provides a channel that the adversary could use to direct the UAV to any available waypoint. For this attack, the input was done by opening a back channel that the attacker could remotely connect to in order to direct the UAV to a specific waypoint.

3.3.1.2 Raspberry Pi

The Raspberry Pi, shown in Figure 26, is a 3.4" X 2.2" X 0.8" SBC which has gained popularity because of its affordability (approximately $35). This 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 which is responsible for bytewise 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.

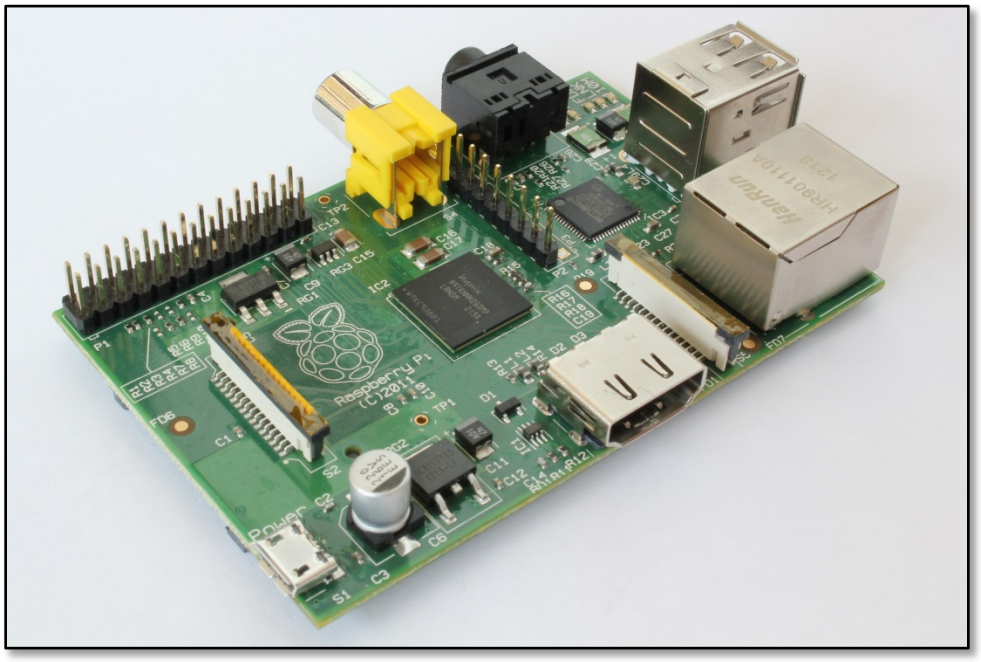


Figure 26. Raspberry Pi SBC.

3.3.1.3 SiCore SHIELD 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 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.

The actual implementation and fabrication of the SHIELD card that is being made to support the Sentinel for the UAV will be accomplished during Phase 2 of this project and will be outlined in section 4.2.2. That section will also detail the design decisions that were made during Phase 1. By using the SiCore SHIELD card, we will be adding additional potential security features to the Sentinel including protections of data bitstream, securing storage, 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.

3.3.1.3.1 Original Card Overview

The SHIELD Coprocessor was designed and developed by Sicore Technologies. It provides a secure enclave to store mission critical data and a secure framework to run mission critical applications. The coprocessor is protected by anti-tamper circuitry that includes a conductive mesh, temperature sensors, and voltage sensors. Tripping the anti-tamper circuitry causes all keys stored in the secure enclave to be zeroized.

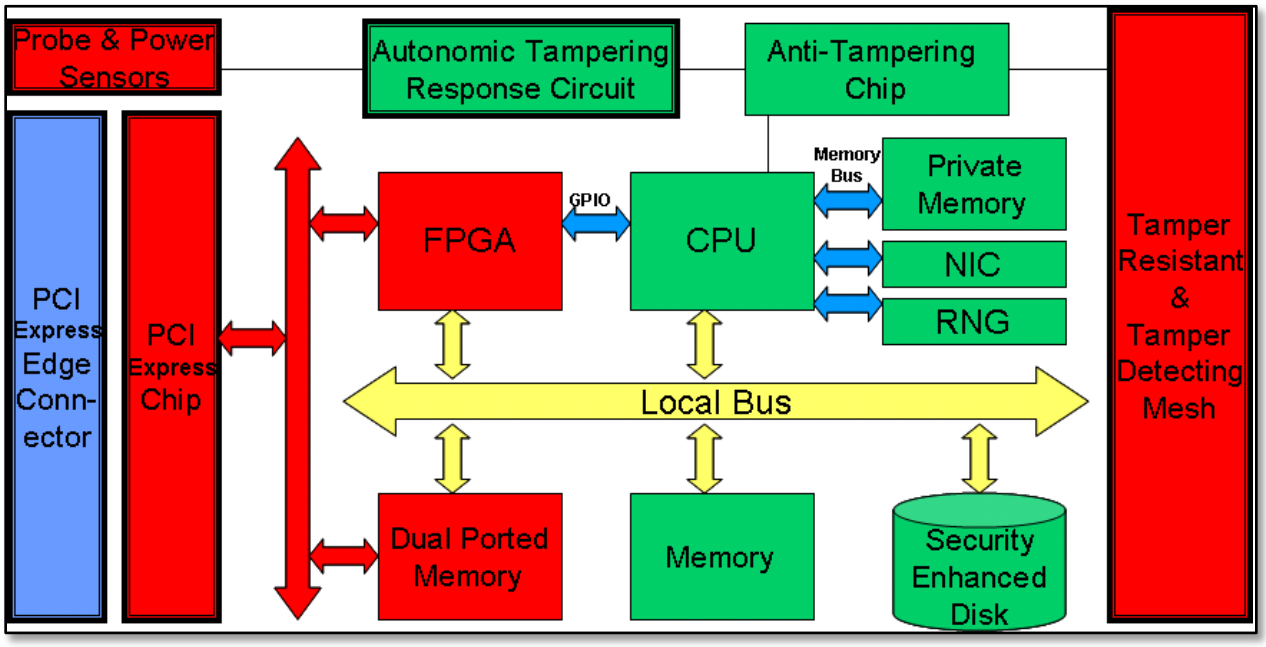


Figure 27. Block diagram of the Sicore SHIELD Coprocessor.

3.3.1.3.1.1 PowerPC 460EXr processor

The PPC460EXr processor runs SHIELD's secure framework called the Module Foundation Firmware (MFF). The MFF is stored in flash memory accessible to PPC. One sector of flash is write-locked through a hardware mechanism. The Module Foundation Firmware

- Initializes hardware on the coprocessor
- Ensures the integrity of programmable hardware on the coprocessor using the SHA256 cryptographic checksum
- Encrypts and Decrypts commands and flash memory with the AES algorithm (256 bit EBC)
- Authenticates and verifies commands from an administrator using the RSA Algorithm (4096 bit)
- Manages users and their applications
- Loads and runs critical applications as specified by a user

The PPC runs the cryptographic algorithms, but it gets the keys for the algorithms from the secure microcontroller through the FPGA.

3.3.1.3.1.2 MAXQ1103 Secure Microcontroller

The MAXQ1103 Secure Microcontroller acts as the secure enclave for the coprocessor's cryptographic keys. All critical software components are encrypted by keys stored in the MAXQ's zeroizable memory. In a tamper event, the keys are zeroized, making all data encrypted by them inaccessible. The MAXQ stores the hash of the Module Foundation Firmware as well and will not release keys to a corrupted MFF.

Communication between the MAXQ1103 and PPC is enabled through a Cyclone II FPGA, which acts as a conduit between the two processors that must be masters of their buses.

3.3.1.3.1.3 Additional Specifications

- 512MB DDR Memory
- 64GB Flash
- 2x Ethernet Ports
- 1x SATA Port
- PCIE 4x
- Cyclone II FPGA
- Cyclone III FPGA

3.3.2 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 28, 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 her 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 herself! A special cyber security officer can make facilitate our ability to address the attack without alerting the insider.

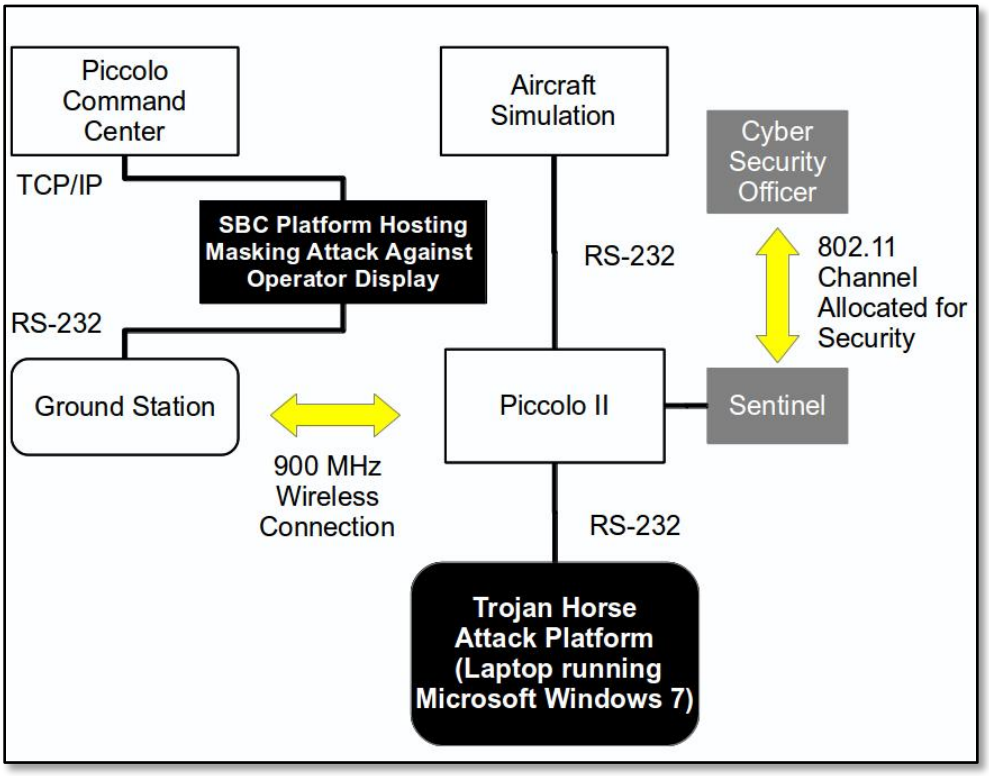


Figure 28. Super secure smart Sentinel for protection with a designated cyber security officer.

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 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. As seen in Figure 29, for the initial prototype this security back channel is represented as a secure 802.11 network.

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 prototype 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.

For the initial prototype system the CloudShield was selected to provide the Sentinel functionality for monitoring the Piccolo autopilot system. The CloudShield was selected as the prototype Sentinel to protect against parameter-based attacks for its rapid reconfigurability and its deep packet inspection capabilities. See section 3.3.1.1 for a more detailed description of the CloudShield platform. Figure 29 shows the architecture used to protect the Piccolo autopilot.

As discussed in section XXX, an interface for the cyber commander was for the created bench-top prototype that provided two functions:

1. The capability to receive alerts about (un)authorized changed to the flight plan.
2. The ability to restore the original flight plan when a cyber attack has been detected.

While this has been valuable for rapid prototyping, an interface for the cyber commander with a more robust feature set is desirable:

* Capability to display position of multiple aircraft in near real-time to the cyber commander.
* Capacity to log information to facilitate analysis, defense, restoration, and forensics.

* Ability to support human-in-the-loop experimentation.
* Developed in an environment to enhance its portability.

As a first step, we have begun the process of migrating the current cyber commander functionality to LAMP (Linux, Apache web server, MySQL, Python) stack. This would provide a structured way to store, retrieve, and analyze information, facilitate portability, and enable us to leverage COTS mapping technology. Currently we have migrated the functionality into a web environment using Django; this includes the capability to show multiple aircraft to the cyber commander using the Google Maps API. Furthermore, cyber detections and a history of each aircraft's flight can be stored for analysis.

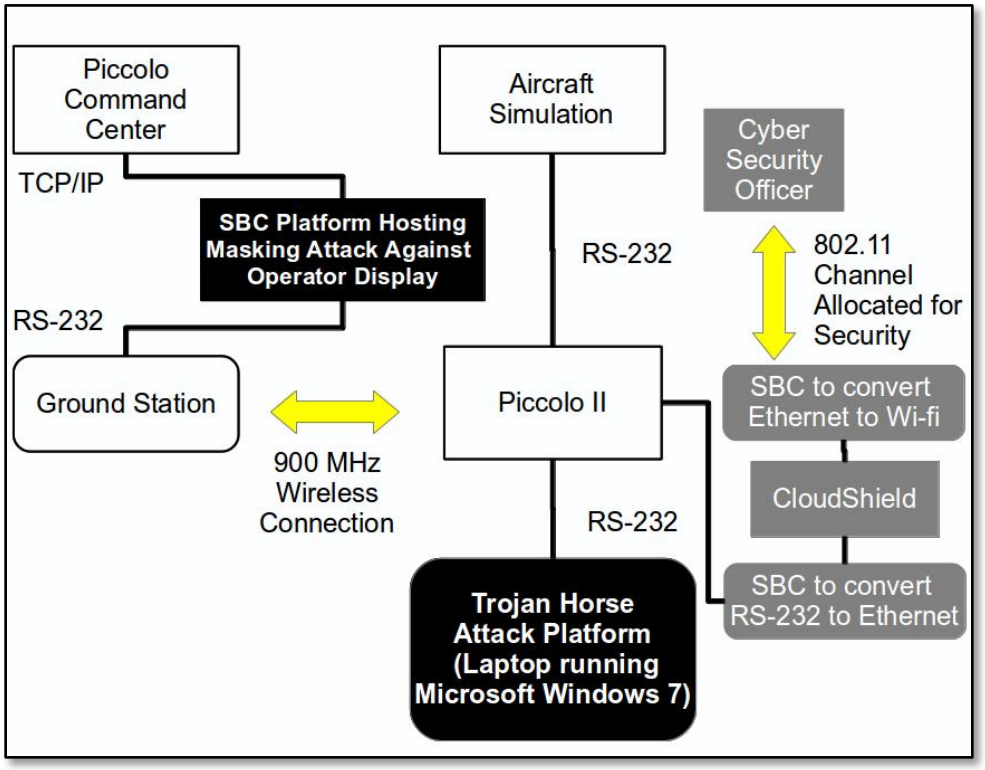


Figure 29. Sentinel (CloudShield) using SBC to convert data from RS-232 to TCP/IP.

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. For the prototype Sentinel, the CloudShield was also selected. As the number of CloudShield CS-2000 units available was limited to one, the same CloudShield served as the Sentinel for the PCC as for the Piccolo autopilot. The final architecture is shown in Figure 30.

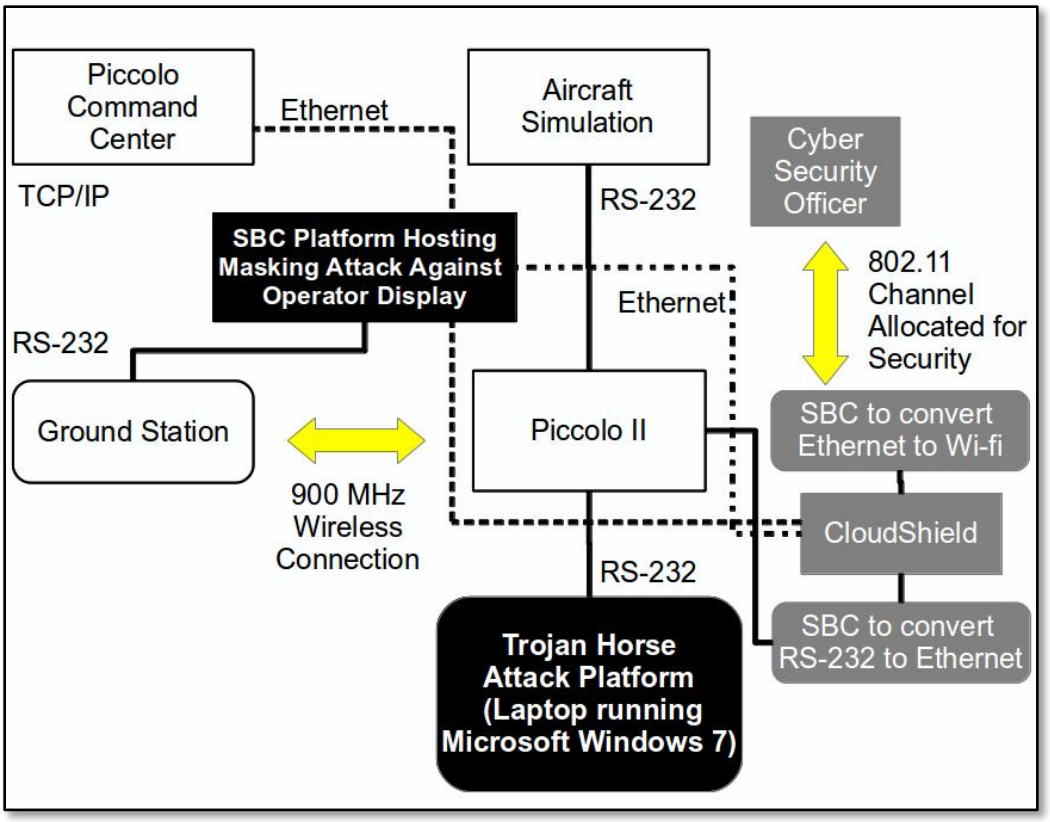


Figure 30. CloudShield Sentinel monitors Piccolo Command Center interface for integrity violations.

As a first step towards transitioning the work developed on the bench-top environment to a flight-ready Sentinel shown in Figure 31, our team began by migrating the functionality of the CloudShield to single compute boards (specifically the Raspberry Pi). In addition, the bench-top prototype utilized an 802.11 network to provide a secure communications backchannel for communicating security related information and commands from the cyber commander to the Sentinel protecting the Piccolo autopilot system. However, in order to ensure flight safety, all security communications will be transmitted using the Piccolo's onboard radio. Thus, part of the initial migration includes modifications to send and receive security information and commands related to Sentinel over the Piccolo's onboard radio. An interim step which utilizes a separate radio used for sensor payload services may be utilized as the full communications mechanism through the Piccolo’s onboard radio is developed for flight demonstrations.

Figure 31 shows the state of our miniaturization efforts. This includes:

- The partial migration of the Sentinel's functionality from the CloudShield to the single compute boards.

Figure 31 – Early Prototype of Flight-Ready Sentinel

- Migration of the onboard status monitoring for (un)authorized changes to the UAVs flight plan.
- Migration of the ground station status monitoring for (un)authorized changes to the UAVs flight plan.
- Sentinel used for protecting the Piccolo autopilot sends security information through the Piccolo's radio to the Sentinel protecting the ground stations that forwards that information to the cyber commander for analysis. To keep this information segregated from the information sent for normal this is done using the PAYLOAD_STREAM provided by the Piccolo autopilot for sending user defined information.
- Communication from the ground station Sentinel to the cyber commander has been changed from wired Ethernet to an 802.11 (b/g/n) network.

Future miniaturization efforts will include restoration capabilities (Section XXX) and the security measures meant to enhance the security of Sentinel. Also, to allow the migration of Sentinel functionality from the CloudShield and the development of the Sentinel SHIELD card to occur in parallel, the current state of miniaturization uses a single compute board to provide conversion from RS-232 to TCP/IP.

3.3.3 GPS System Attack Detection and Mitigation

Before we can describe our Phase 1 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 propose an architecture of several stand-alone navigation systems (i.e., GPS INS, and the Piccolo navigation system). We plan on housing these extra redundant navigation systems in the Sentinel. 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 a secondary INS and GPS units. These units would be supplied by a vendor different than those embedded into the Piccolo and would connect directly to the Sentinel. The redundant INS should be able to produce the same sort of procedures compared to the Piccolo II's internal INS. We will use these components to verify the behavior of each component to see if one or more of these components are performing anomalously.

In order for an adversary to successfully exploit a UAV navigation system, she 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.

Frame of Discernment

The purpose of the Frame of Discernment (FOD) is to enumerate the exhaustive and mutually exclusive scenarios. For our purposes, Table 10 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 10. Frame of Discernment for Navigation Architecture.

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 $\mathrm{m_{1,t}}$ is 1 (for the case $\mathrm{x = y = z}$), 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 5.1 at the end of this report.

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 $2^{3+\mathrm{N}}$ events. In effect, the mass functions have to adapt to the additional N components.

For example, if $\mathrm{N} = 2$, 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 11. 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.

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

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

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 $\hat{m}_j(t)$ 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 12. 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 13. 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

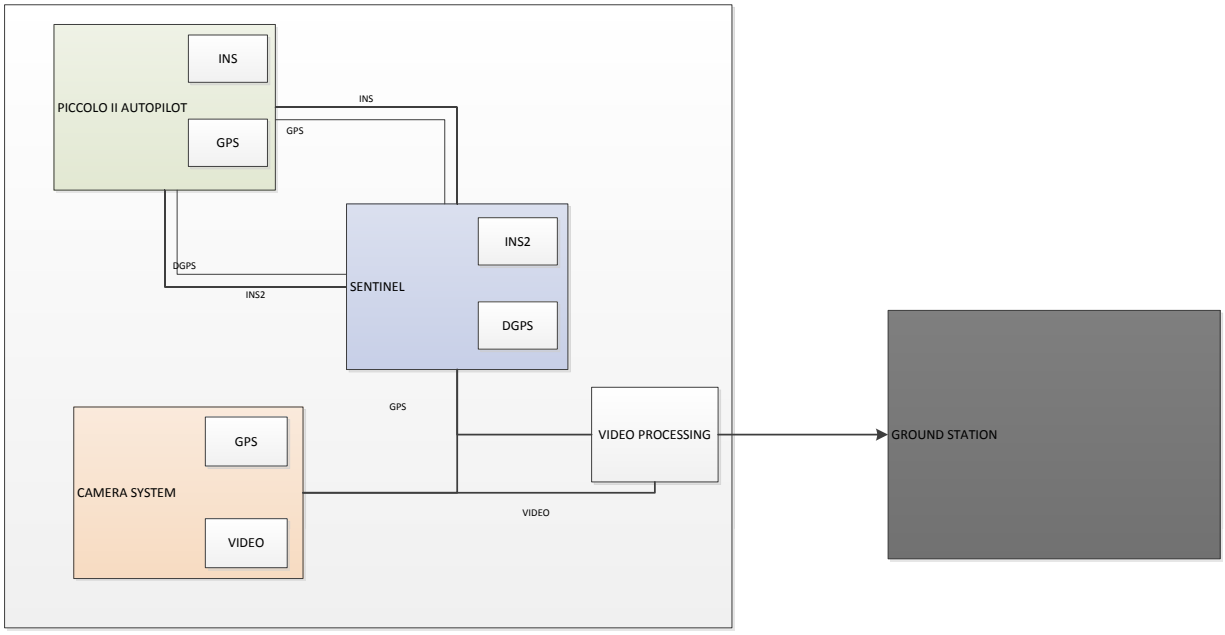


Figure 32. Architecture for camera system.

Figure 32 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—GPS and INS 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 14 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 14. 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

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

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 15. Procedures for camera system.

Experimental Design for Testing Detections in Multiple GPS /INS Signal Configurations

We have designed and initiated tests of a series of ground-based experiments to investigate methods for detecting deviations and anomalous sensors, and potential attacks on GPS and INS sensors on the Piccolo and the external GPS sensors that will be introduced as part of the Phase 2 Sentinel implementation. Adversaries can target these sensors by manipulating their measurement outputs and relay falsified data that are ultimately fed back to their corresponding controllers.

Methods

Diverse Redundant Components. We are experimenting with an architecture of several stand-alone navigation systems. A diverse redundant component system is, in this case, a system comprised of multiple components with the same purpose. The aim of using multiple components is to avoid downtime after a successful attack and force the attacker to manipulate multiple components. In our proposed phase 2 design, we will equip the aircraft with the principle navigation system (Piccolo II Autopilot) and two redundant components (a GPS and an INS).

We design these components to work in tandem such that if one component is compromised the redundant components could succeed in restoring the system and avoid downtimes.

Redundant components also increase the difficulty of success. In order for an adversary to successfully attack a diverse redundant system increases the difficulty of a successful attack by forcing an attacker to manipulate multiple components simultaneously. Otherwise, if the adversaries control only one or a few of these sensors, managers would be able to detect the intrusion and isolate the problem with relative

ease. Component architecture with multiple components increases the cost of success for the adversaries.

Frame of Discernment. Suppose we have N redundant sensor components, and each component can hold one of two mutually exclusive characteristic: $\mathrm{Faulty} = 0$ OR $\mathrm{Reputable} = 1$. Let X be the universal set: the set representing all possible states of a system under consideration. The power set 2^{N} is all the set of all subsets of X, including the empty set $\emptyset$.

Using a specific example, suppose we have $\mathrm{N} = 3$ sensors $\{s_1, s_2, s_3\}$. The system has $2^3 = 8$ mutually exclusive proposition. We use similarity measurements to discern which proposition is the most likely occurring.

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

Figure 33 Frame of Discernment (FOD) of 3 diverse redundant sensors for UAV navigation

For the UAV, we install 3 redundant components (See Figure 33), and each component can hold one of two mutually exclusive characteristic: $\mathrm{Faulty} = 0$ OR $\mathrm{Reputable} = 1$. In Figure 33, Event 1 indicates that all components are functioning reliably; while Event 2 indicates that all components but the Piccolo II navigation system is functioning reliably.

Subjective Logic. Since we cannot directly observe whether or not a sensor is faulty or reputable, we rely on the evidence taken from the 3 sensors. We can apply methods in Subjective Logic. We apply a detection scheme based on a subjective binary logic framework. A fundamental aspect of security is that nobody can ever determine with absolute certainty whether a proposition about a component is true or false. We do not know whether or not a component is attacked or it could merely be a systematic fault. We can only make inferences based on the components' observable behavior.

An opinion is denoted by ω_{x} where x is the proposition in the FOD to which the opinion applies. We assess each proposition by four characteristics 1) belief that the specified proposition is true, 2) belief that the specified proposition is false, 3) amount of uncommitted belief (uncertainty), 4) a priori probability of the event happening.

Let x be a proposition. A binomial opinion about the truth of x is the ordered quadruple $\omega_{\mathrm{x}} = (\mathrm{b}, \mathrm{d}, \mathrm{u}, \mathrm{a})$ where:

1. $\mathrm{b} \in [0, 1]$: belief that the specified proposition is true; an example of absolute opinion,
2. $\mathrm{d} \in [0, 1]$: belief that the specified proposition is false; another example of absolute opinion,
3. $\mathrm{u} \in [0, 1]$: amount of uncommitted belief; uncertainty
4. $\mathrm{a} \in [0, 1]$: priori probability in the absence of evidence

These components satisfy the following properties:

$$b + d + u = 1$$

An opinion where $\mathrm{b} = 1$ is indicating that the binary logic is true while $\mathrm{d} = 1$ indicates that the binary logic is false. And $\mathrm{b} + \mathrm{d} = 0$ indicates total uncertainty.

We tie in the concept of similarity measurement procedures with each proposition in the FOD. There are families of viable similarity measurements. In this analysis, we chose to use the general L_1, "City Block," similarity measurement. For our purposes, 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 diverse sensor measurement with one another.

Define Gaussian random variables $\mathrm{X_t}$, $\mathrm{Y_t}$, $\mathrm{Z_t}$ represent a multivariate values of the 3 sensory components 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 the belief function, b, for Proposition 1:

$$b_{1,t}(X_t = x, Y_t = y, Z_t = z) = \left|\frac{x_i - y_i}{\sigma_{xy}}\right| \left|\frac{x_i - z_i}{\sigma_{xz}}\right| \left|\frac{y_i - z_i}{\sigma_{yz}}\right|$$

And the *disbelief* function for Proposition 1:

$$d_{1,t}(X_t = x, Y_t = y, Z_t = z) = \left|1 - \left(\frac{x_i - y_i}{\sigma_{xy}}\right)\right| \left|1 - \left(\frac{x_i - z_i}{\sigma_{xz}}\right)\right| \left|1 - \left(\frac{y_i - z_i}{\sigma_{yz}}\right)\right|$$

We allow the cyber security operators to modify σ_{nm}based on the current mission. The cyber security operators provide us with two parameters: 1) the *maximum acceptable deviation* and 2) the *minimum unacceptable deviation.* From there, we can deduce the values of σ_{nm}. We believe that providing the operators with σ_{nm} this gives the operator more control for the allowable deviation.

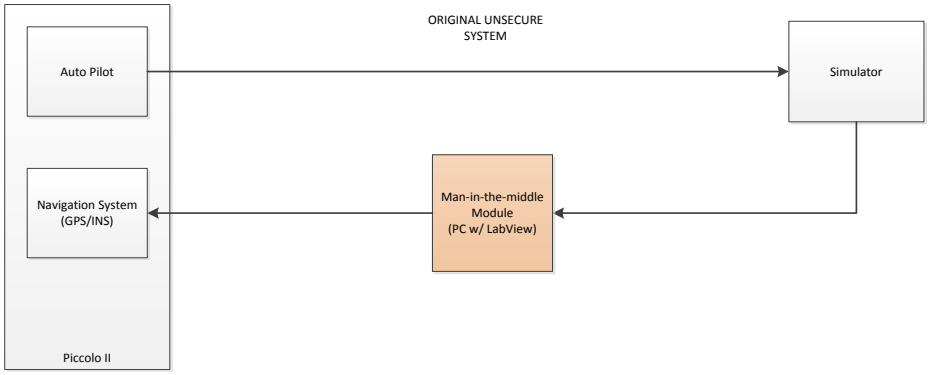
The belief functions will also take into account the geometry of the earth. We will modify the belief function to more accurately reflect the elliptical/spherical shape of the Earth. The idea is that 1 degree difference in longitude near the equator is much farther than the 1 degree difference in longitude near the poles. We will borrow ideas from non-Euclidean geometry to find the distance between two coordinates. See Section 5.2 for example code for this calculation.

Different navigation components have different sampling rates which require us to interpolate data from and between two sequential data points. In this project we use a linear interpolation technique as we need to avoid filtering signals using the data coming from alternative components. See Section 5.2 for example code for this calculation.

Flight Simulation Experiment

To simulate the detection abilities of the algorithm, we inject the autopilot with false coordinates. Below is the standard interaction of the autopilot and the simulator. The actuating commands is fed into the Simulator which feeds back to the autopilot the estimated coordinates during a flight. The simulation

system is configured with an additional “Man-in-the-middle” module which enables us to alter the true coordinates the simulator feeds into the Navigation System.



3.3.3.1 Fig. 2 Implementation of the Man-in-the-middle Module within the simulation environment

During the Flight Simulation Experiment, we will gradually change the coordinates of the aircraft and collect the coordinates. We test 4 different longitude deviation rates:

1. 0 micro second degree deviation per 1 second
2. 4 micro second degree deviation per 1 second
3. 8 micro second degree deviation per 1 second
4. 16 micro second degree deviation per 1 second

During this experiment we would like to see how much an adversary would deviate the aircraft without detection.

Ground-based Field Experiment

A diversely, redundant navigation sensors is a key component to the architecture. For our supplemental navigation device we decided on the Adafruit Ultimate GPS (See Appendix for Specifications and Properties). To test the algorithm, we designed a field experiment which we used two ground-based vehicles to gradually deviate from one another. We recorded GPS output using an attached Linux based EEE PC netbook.

We compare output location from two Adafruit GPSs to that of the Piccolo II GPS. We drew Latitude and Longitude information using a Piccolo Command Center plug-in from the *GPS_HEALTH* packet, which updates every six seconds.

NOTE: Initial testing was discarded due to the noise coming from the INS within the *TELEMETRY_LO_RES.* Testing the *GPS_HEALTH* data will commence shortly.

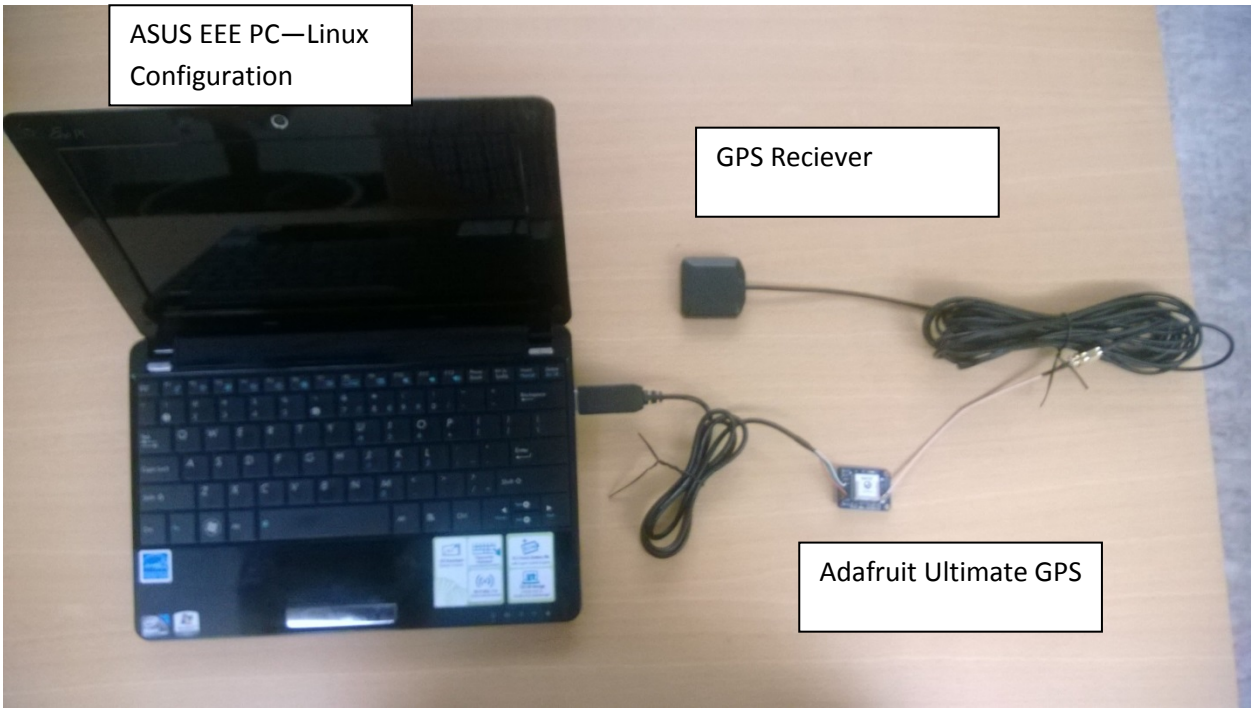


Figure 34 – Ground-based Navigation Experimental Environment

We have designed the field experiment to have 5 different types of deviations:

1. No deviation—the two vehicles' paths are equal
2. "Small" deviation rate
3. "Medium" deviation rate
4. "Large" deviation rate
5. "Instant" deviation rate

Results of these tests will be used to test the fidelity, judge the effects of noise and timing issues related to this approach, and to fine-tune the models in their associated sensitivities for detecting deviations and classifying the detections as attacks that will be used in the in-flight versions of these models.

3.3.4 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.

3.3.4.1.1 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 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.

3.3.4.1.1.1 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.

3.3.4.1.1.2 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.

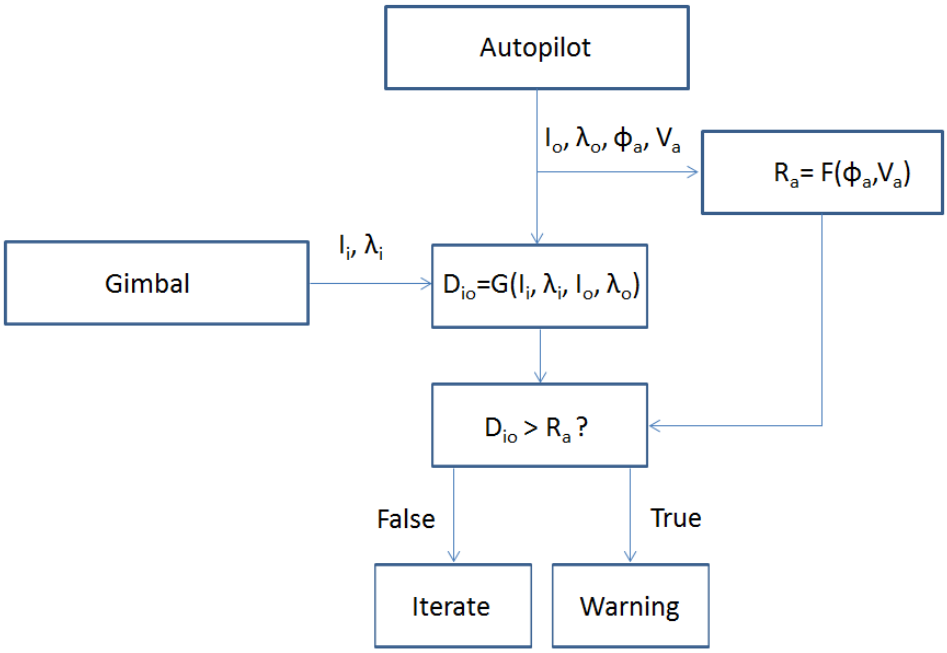
The aircraft velocity and position can be determined by monitoring the gimbal telemetry stream for packets of type HOST_GPS_DATA_GIMBAL_PKTTYPE and of group GIMBAL_POSITION_INFORMATION_GROUP. These telemetry packets can be decoded to give the GPS position and velocity of the aircraft. These two pieces 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.

3.3.4.1.1.3 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 3.3.4.1.1.2, 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 3.3.4.1.1.1, 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.

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 (ϕ_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. Preliminary code has been written to implement these calculations.

An approximation of this WGS84 model using the haversine formula for a spherical Earth is represented here:

$$haversin\left(\frac{d}{r}\right) = haversin(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\, haversin(\lambda_2 - \lambda_1)$$

where the haversin function is defined as:

$$haversin(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

$$D = R\, haversin^{-1}(h) = 2R \arcsin(\sqrt{h})$$

$$D = 2R \arcsin\left(\sqrt{haversin(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\, haversin(\lambda_2 - \lambda_1)}\right)$$

Where ϕ_1 and ϕ_2 are the latitudes of point one and two, and λ_1 and λ_2 are the longitudes of point one and two. R is the radius of the spherical Earth approximation. Soon this great circle distance computation will be replaced by the WGS-84 model using an ellipsoidal Earth.

3.3.5 Hardware Security Against Design and Manufacturing Attacks

3.3.5.1 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.

3.3.5.2 Detection/restoration of hardware Trojans

Phase 1 focused on preliminary work to investigate the capability of TMR as a hardware security technique. Figure 35 shows a block diagram of a TMR design with three cores; each core has its own clock signal, cache and memory.

In the test design, the input signal is a 7-bit general-purpose input/output (GPIO) input which is connected to the switches on the FPGA board. The software run by the cores inverts the 7-bit input and

sends the inverted value to GPIO output. Since the three cores run with different clock frequencies, synchronization is required.

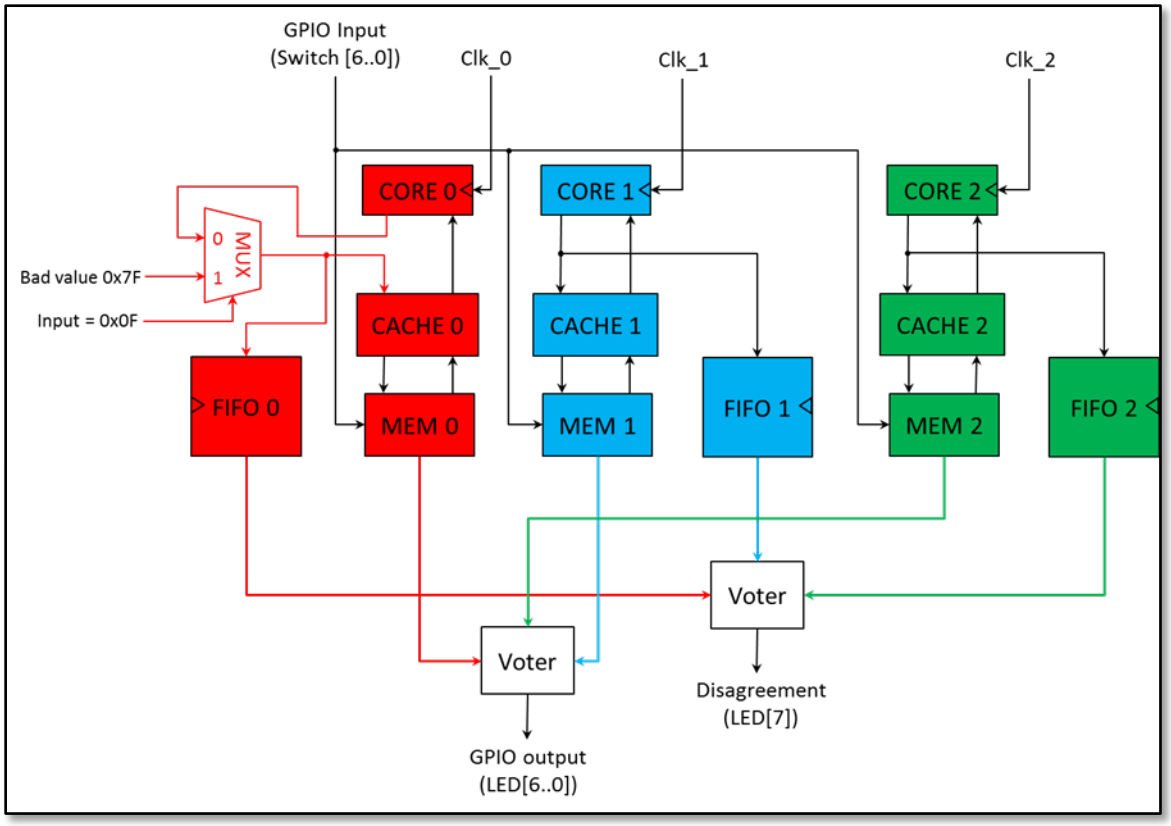


Figure 35. Block diagram of a FPGA configured to provide detection and recovery with TMR.

Synchronization is implemented by adding a first-in-first-out (FIFO) buffer to each of the processor. The data written into each cache is also written into the corresponding FIFO simultaneously. When a FIFO is empty its corresponding processor runs as normal. When a FIFO is not empty (i.e., a value has been written into the cache) the processor halts and waits for the same instruction to be executed on other processors. When all three FIFOs have a value a read operation is performed and the values are sent to a voter. If the values do not agree then an LED is asserted to indicate the disagreement. Another voter is used to determine the GPIO output which is connected to the LEDs on the FPGA board.

In Figure 35, an example hardware Trojan is inserted into core 0. If the 7-bit GPIO input is 0x0F then the data written into the cache is 0x7F; i.e., it has been tampered with. In all the other cases, the cache always gets the correct value. This mimics a piece of malicious hardware that is triggered by a specific input. When the Trojan is activated it modifies data and results in incorrect output. With the TMR design shown in Figure 35 this malicious operation can be detected and masked, assuming the other two processors are operating correctly.

Figure 36 shows the operation of the TMR design. The lower 7-bit LEDs are GPIO input, which is 0x0F, and therefore triggers the hardware Trojan. In this situation, the data written into cache 0 is changed to

0x7F, and the output of core 0 is therefore 0x00 (0x7F inverted). But the TMR design is able to give the correct output, 0x70, in spite of the malicious operation in core 0. Meanwhile, the highest LED is asserted to indicate that a disagreement is observed among the three processors.

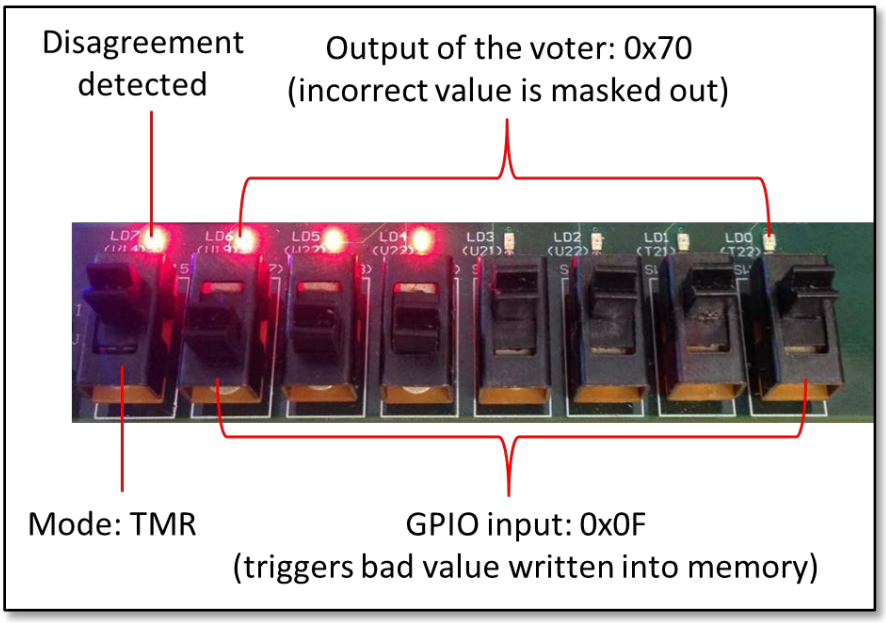


Figure 36. TMR Detects and Masks Deviated Operation.

The idea of monitoring memory accesses can be applied to detect the attacks specific to protocol conversion which are described in section 3.2.4, because any meaningful changes in the data flow will eventually be seen in memory activities.

3.3.5.3 Authentication and Verification of FPGA bitstreams and ICs

For more general attacks in the supply chain, such as reverse engineering, tampering with the design and replacement of FPGA devices, encryption and authentication of the FPGA bitstream is required.

Encryption can be used to protect a FPGA bitstream from being reverse engineered or modified. For example, Advanced Encryption Standard (AES) uses a private key which is kept secret to both encrypt and decrypt the data. The designer generates the private key and uses it to encrypt the bitstream. The private key is also loaded into the target device and is used for decryption when the user loads the bitstream to the FPGA. The vulnerability of using a single secret key is that the encryption could be broken once the key is stolen. Since the private key is pre-loaded in the FPGA it is possible for an attacker to obtain the key if he has physical access to the device. If the private key cannot be kept secret then a successful decryption does not guarantee the original bitstream provided by the designer.

On the other hand, public key authentication mechanisms can provide a sign-and-verify process to ensure security. For example, the RSA algorithm uses a private key and public key; the private key being secret, and the public key being open. The two keys are mathematically linked, but it is impossible to calculate the private key using only information known about the public key. The designer encrypts the

FPGA bitstream with his private key (also known as signing) and loads his public key in the FPGA device. With the pre-loaded public key the user can decrypt the bitstream (also known as verification). Assuming the private key is kept secret a successful decryption indicates that the bitstream is the one signed by the designer. In authentication the private key is kept only by the designer; thus, it is much safer than the single private key used in encryption.

Private key cryptography can provide protection against reverse engineering, but does not provide authentication. Public key cryptography can verify an authorized message, but the message can be viewed by anyone who has the public key, and therefore it cannot protect the IP information.

Another possible attack is that the chip, module, or board is substituted during or after manufacturing. Detection of this kind of attack requires some form of a *circuit signature*. For example, physically unclonable functions (PUF) can provides such a *signature* or *fingerprint*. When a physical stimulus is applied to a structure it reacts in an unpredictable (but repeatable) way due to the interaction of the stimulus with the physical microstructure of the device. This microstructure depends on physical factors introduced during manufacture which are unpredictable. The applied stimulus is called the challenge, and the reaction of the PUF is called the response. A specific challenge and its corresponding response together form a challenge-response pair. The device's identity is established by the properties of the microstructure itself. The selection of an appropriate PUF as a circuit protection mechanism is part of our long-term future work.

Testing of Prototype Hardware Design on NetFPGA card

The initial work on testing the prototyping efforts during phase I for the protections on the for the Hardware Security Against Design and Manufacturing Attacks has been developed and tested using the KC705 development card. SiCore has provided a prototype card for use in the project to aid in testing both the logical implementation of the protections and design patterns and to verify the hardware configuration of the designs for the card that will be manufactured for the Sentinel by SiCore. The table below describes the differences between the initial development environment and the prototype board provided for testing by SiCore.

	KC705	**NetFPGA**
User FPGA	Kintex-7 XC7K325T-2FFG900C	Kintex-7 XC7K325T-1FFG676
Reset	Active high	Active low
On-board I/O	4 switches, 5 buttons, and 8 LEDs	4 buttons and 4 LEDs
UART Interface	With flow control signals	No flow control signals
Ethernet Interface	1 Marvell Alaska 88E1111 PHY (MII/GMII/RGMII/SGMII)	4 Realtek RTL8211 PHYs (RGMII only)

Table 16 – Differences between Kc705 and NetFPGA development environments

The cores that have been instantiated on for the development of the hardware security protections have been tested using the LEON-3 soft core CPU on the FPGA onboard the KC705. The functionality that has been developed has included sending data over the UART to the LEON-3 CPU, incorporating the GPIO functionality of the board in the tests for functionality verification, accessing the external DDR-3 and the on-chip BRAM as memory for the tests, creating a preliminary TMR design using the on-chip

BRAM. In addition, the development work has been extended to include the UART receiving data from the CPU and the Ethernet adapter sending data. These efforts were initially implemented using the KC750 environment.

In recent efforts, we have endeavored to move the functionality of the initial designs over to the NetFPGA environment. This has included successfully instantiating the LEON-3 soft core on the NetFPGA board, resetting polarity changed from active-high to active-low, sending data from the UART, receiving data by the UART, testing the GPIO functions (buttons, LEDs) and testing the use of on-chip BRAMs as memory. We are currently evaluating the ability to send and receive data through the Ethernet adapter, establishing TCP/IP streams to the Raspberry Pi portions of the Sentinel. In addition, we are investigating the implementation of the TMR designs and verifiable voting schemes in both the KC705 and NetFPGA test environments.

Side Thoughts of TMR and Bitstream Authentication

The design shown in Figure 35 is for the purpose of demonstration. In a real design the voter should be put before the memory: the three copies of data written to memory are compared and voted, and the voting result is written to the three copies of memory. This ensures that any corrupted data will be corrected before being stored and that all of the three cores have a correct copy of data.

A UAV requires real-time operations, and as such so should the detection and restoration processes used to protect critical systems. The voter of a TMR module will increase the delay of the data path to memory or output signals, and may result in an extra clock cycle if the original clock speed cannot be met. This slight change in timing should not affect the real-time requirement.

When a mismatch is detected in a system using TMR it could be the result of either a persistent mismatch or a transient mismatch. A persistent mismatch is most likely to be caused by a permanent fault or a cyber-attack such as malicious software and/or hardware; while a transient mismatch is most likely to be caused by a single event upset (SEU). A transient fault should not last longer than the period of a single instruction. Therefore, any mismatch longer than that should be considered as a persistent one. A transient mismatch can be safely ignored, but a persistent mismatch must assert a warning to the human operator indicating that the flagged core needs investigation or repair.

By definition, a system with TMR can tolerate up to one failed module. When a persistent mismatch is detected it is appealing to abandon the corrupted core. However, this makes the rest of the design vulnerable to transient faults caused by SEUs, because a SEU can still occur in the functional cores. In this case, the hardware must be replaced as soon as possible. In the meantime, if an error is detected (i.e., when a majority vote is not available) the system should enter a fail-safe mode. Alternatively, if hardware support allows, it can roll back and retry.

Although TMR is an effective technique for security, it has its own Achilles' heel; i.e., the voter. Since the voter is not triplicated, it is vulnerable to attacks. Two options are available:

1. Put the voter on a piece of trusted hardware which is kept separated from the FPGA.

2. Keep the voter in the FPGA design and leave the security concern to the FPGA bitstream.

Option (1) is promising if the design is transferred to discrete integrated circuits (ICs) in the future, and is also useful for checking other functions on the SHIELD Coprocessor. In addition, the design of a voter is straightforward, which makes it reasonable to put it on a piece of dedicated hardware. However, this may lead to timing issues because data has to be sent off-chip to the voter and then sent back on-chip again. Furthermore, the dedicated voter must be authenticated or trusted otherwise it would become a single point of failure.

Option (2) is suitable for a FPGA design and other single-chip solutions. The authentication of the voter can be done along with other FPGA designs. These tradeoffs need to be considered when making the design decisions.

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 design-related questions that can support future efforts related to implementations of the System Aware Cyber Security concept. 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 the ongoing project transitions from hardware-based simulation to a flight-ready hardware implementation of the Sentinel, we will need to continue to refine the answers to these questions and develop the methods to collect the data necessary to support evaluations of the Sentinel solutions. Though the focus of this prototyping effort is focused on the specific Outlaw ER aircraft surveillance system configuration and the Piccolo autopilot, these questions also need to be addressed in the framework of how the Sentinel functionality would answer them in a more generic class of physical systems.

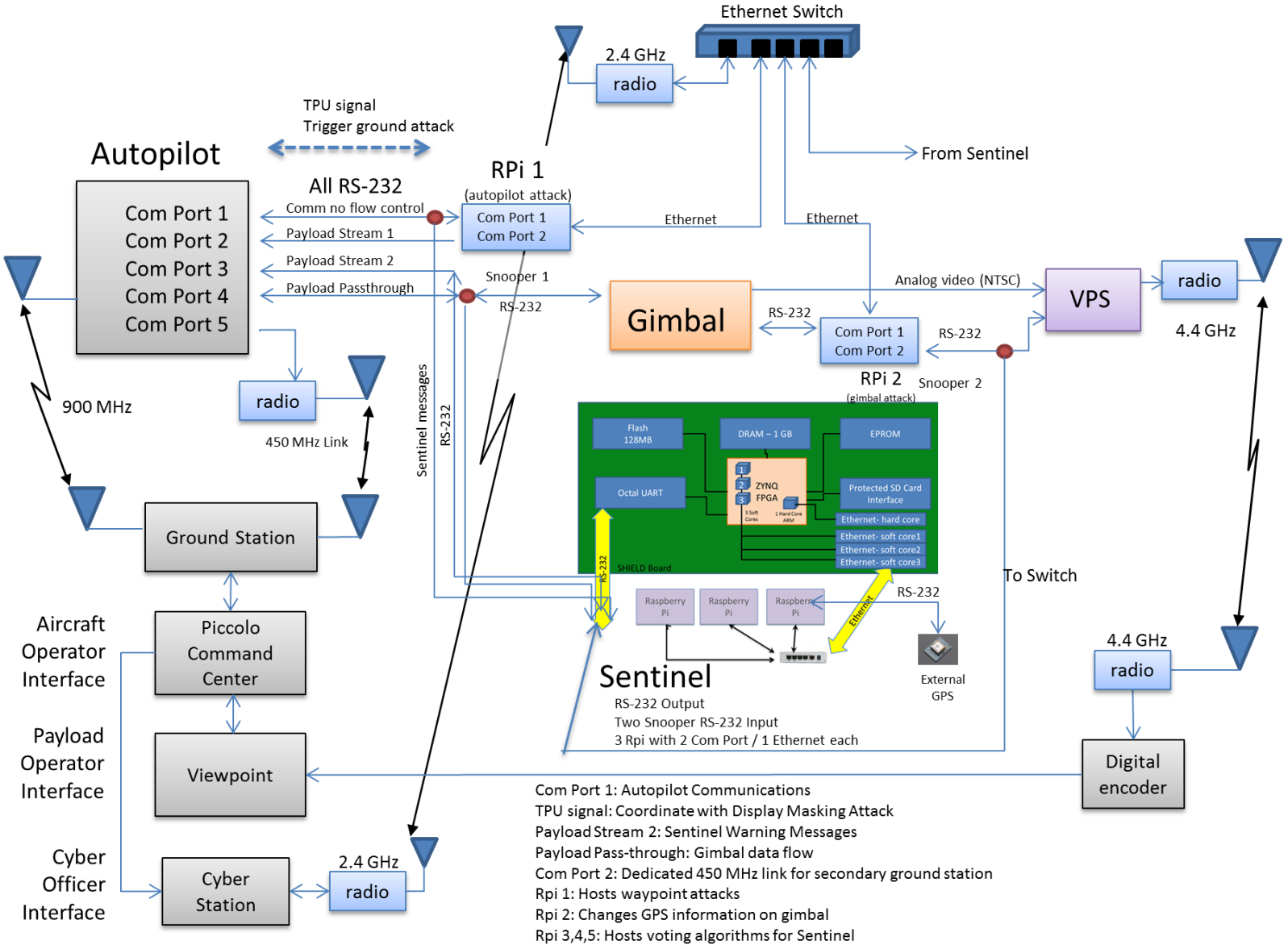
4 Proposed Work for Phase II

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 noted section 3.3.1.1, the CloudShield does not meet the weight and size requirements necessary for airborne use. As a result, a flightworthy Sentinel will be developed based on the SiCore SHIELD CoProcessor discussed in section 3.3.1.3. The Sentinel algorithms for detecting the attack, alerting the operator, and taking remedial action will be based upon those developed in RT-42. The following subsections describe the tasks that will be performed in order to conduct a flight demonstration of the Sentinel concept.

4.1 Proposed Hardware Architecture for Flight Demonstration

Figure 37 presents the proposed system architecture for the flight demonstration. In the upper left corner of the figure is the Piccolo autopilot. The autopilot communicates with the ground control station through redundant radio links at 900 MHz and 450 MHz. These links are used to send command and control signals from the ground station to the aircraft, as well as send telemetry data from the aircraft to the ground control station. Two of the communications ports on the autopilot (Com Port 1 and 2) will be used in the implementation of the autopilot attack (see section 3.1 for details). A Raspberry Pi SBC (RPi 1 in Figure 37) will host the parameter-based exploit developed under RT-42 that will attack the autopilot by changing its waypoints in the mission list. To hide this attack from the UAV operator RPi 1 will also send a trigger signal to malware on the machine hosting the PCC that will hide the change to the waypoints. This signal will be sent over one of the autopilot's Time Processor Units (TPUs) that can be used as discrete signal communication lines. These lines are monitored or set by the autopilot system and their status (high or low) is encoded in the autopilot message stream. Another TPU will be used to send a signal from the malware on the PCC to RPi 1 to signal the start of the display masking attack. In a similar fashion, RPi 2 will host an exploit used to compromise the GPS data sent from the autopilot to the gimbal then on to the ViewPoint software to be used to view the video (see section 3.2.2 for details). RPi 1 and RPi 2 are both connected to the Cyber Station (lower left corner of Figure 37) via Ethernet over a 2.4 GHz radio link. This connection will be used to communicate with the processors and trigger the cyber-attacks during the demonstration.

The UAV Sentinel will be implemented on a SiCore SHIELD CoProcessor as shown in the center of Figure 37. Details of the UAV Sentinel are presented in section 4.2.2. The UAV Sentinel monitors serial data traffic at the locations depicted in Figure 37 using the Raspberry Pi snooper developed under RT-42 (see section 2.2). The Sentinel has two communication paths. The first one is an RS-232 based serial link through Com Port 3 on the autopilot. Using this link the Sentinel can send warnings and alerts to the UAV operator through the second payload message stream. The other communication path is over Ethernet via the 2.4 GHz radio link, the same link used by RPi 1 and RPi 2.


Ethernet Switch
2.4 GHz
radio
TPU signal
Trigger ground attack
Autopilot
From Sentinel
RPi 1
(autopilot attack)
All RS-232
Com Port 1
Com Port 2
Com Port 3
Com Port 4
Com Port 5
Comm no flow control
Payload Stream 1
Payload Stream 2
Payload Passthrough
Com Port 1
Com Port 2
Ethernet
Ethernet
Snooper 1
RS-232
Gimbal
Analog video (NTSC)
VPS
radio
4.4 GHz
RS-232
Com Port 1
Com Port 2
RS-232
RPi 2
(gimbal attack)
Snooper 2
900 MHz
radio
450 MHz Link
Sentinel messages
RS-232
Flash
128MB
DRAM – 1 GB
EPROM
Octal UART
ZYNQ
FPGA
Protected SD Card
Interface
Ethernet- hard core
Ethernet- soft core1
Ethernet- soft core2
Ethernet- soft core3
SHIELD Board
RS-232
Ground Station
To Switch
Raspberry
Pi
Raspberry
Pi
Raspberry
Pi
Ethernet
RS-232
Aircraft
Operator
Interface
Piccolo
Command
Center
Sentinel
External
GPS
4.4 GHz
radio
Payload
Operator
Interface
Viewpoint
Digital
encoder
Cyber
Officer
Interface
Cyber
Station
2.4 GHz
radio
RS-232 Output
Two Snooper RS-232 Input
3 Rpi with 2 Com Port / 1 Ethernet each
Com Port 1: Autopilot Communications
TPU signal: Coordinate with Display Masking Attack
Payload Stream 2: Sentinel Warning Messages
Payload Pass-through: Gimbal data flow
Com Port 2: Dedicated 450 MHz link for secondary ground station
Rpi 1: Hosts waypoint attacks
Rpi 2: Changes GPS information on gimbal
Rpi 3,4,5: Hosts voting algorithms for Sentinel

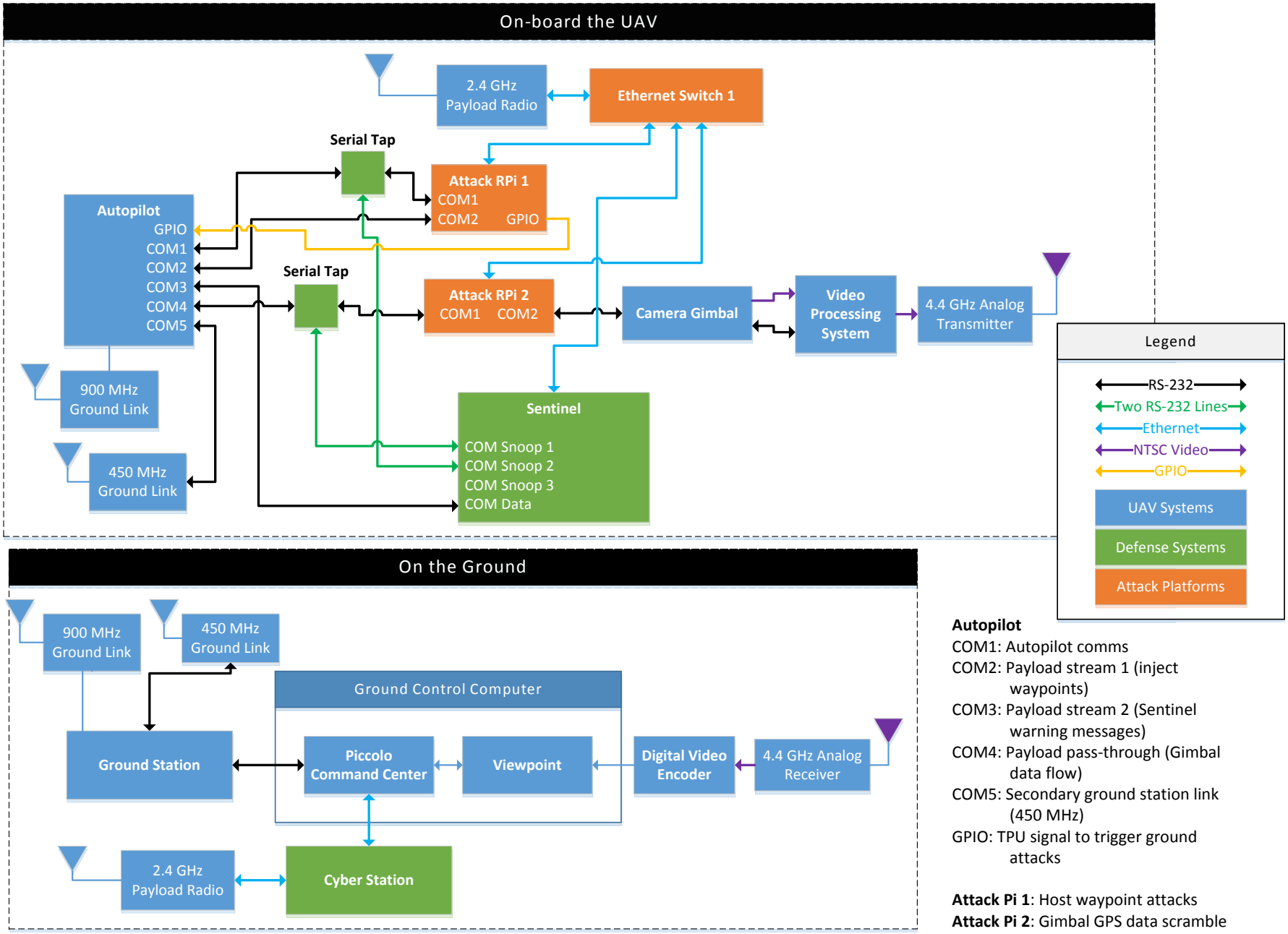


Figure 37. Architectural block diagram illustrating how the UAV SHIELD Sentinel will be integrated into the GTRI's GAUSS platform.

4.2 Needed Development to Reach Next Milestones

4.2.1 Develop on-board and ground attacks

4.2.1.1 Gimbal Attack

Two of the gimbal attacks described in section 3.2.3.3 will be implemented in the flight demonstration. These are the Extend/Retract attack (section 3.2.3.3.1) and the Gimbal Command attack (section 3.2.3.3.3). The source of these attacks will come from malware on the PCC embedded as a plugin. The attack software will monitor the aircraft's location by decoding the telemetry stream and reading the GPS coordinates. The attack will be triggered when the UAV enters a predetermined geographical area.

4.2.1.2 GPS Attack

During the Phase 1 work on exploiting and monitoring the autopilot GPS system, the research team developed algorithms to successfully manipulate GPS data in the HiL emulation environment for the Piccolo II autopilot system and to detect anomalies in the GPS data stream. Unfortunately, this attack utilizes the CAN bus interface, which is only available in the emulation environment. In addition, this attack carries a risk of losing control of the aircraft during flight. However, the algorithms developed for the attack and for the detection of the GPS attack will port directly to the serial interfaces that connect the autopilot to other components on board the plane.

The research team has opted to demonstrate the GPS attack on board the plane by executing an attack against the gimbal GPS instead of the autopilot GPS. The gimbal GPS data is used for locking the gimbal on a point of interest and for geolocating tracked targets. The GPS data from the gimbal is stored as metadata for the video imagery along with other geospatial data, indicating where the images that are sent to the ground for processing or recording into the Cloud Cap ViewPoint software environment were taken.

The GPS attack will be executed by a Raspberry Pi processor RPi 2 onboard the aircraft. The attack will corrupt the GPS data from the gimbal that is stored as metadata with the video. Thus, when the video is viewed it will have incorrect GPS coordinates associated with it. This type of attack is subtle and removes much of the intelligence value from the video both in real time and for forensic purposes (i.e., if the video and its associated metadata is stored for later analytical use). The Sentinel detection algorithms built in Phase 1 will be enhanced to include a third source of GPS data. A separate, external GPS unit will be added to the hardware configuration on board the aircraft which will provide the Sentinel with three sources of GPS data (the gimbal GPS, autopilot GPS and the new external GPS). This will enable the Sentinel to employ diverse redundancy and verifiable voting security design patterns to aid in eliminating corrupted GPS data streams and to provide restorative capabilities to the during the flight demonstration.

4.2.1.3 Parameter-Based System Attack

The goal of this task will be to implement the parameter attack outlined in section 3.2.1 for the flight demonstration. The in-flight demonstration of the parameter-based attack will be focused on attacking

the waypoint parameters for the flight plan; i.e., modifying the flight on the Piccolo autopilot and hiding the attack from the operator by executing a secondary attack on the PCC.

As seen in Figure 37, the platform selected for the exploit is a Raspberry Pi SBC attached to the Piccolo autopilot (RPi1 in Figure 37). To deploy the exploit on a Raspberry Pi will require several modifications to the exploit developed for RT-42:

- Original exploit was developed for a laptop running the Windows 7© OS using a SDK provided by Cloud Cap Technology. The target platform is a Raspberry Pi running the Raspbian (i.e., Linux) OS using an updated version of the communications SDK provided by Cloud Cap Technology. The exploit developed for RT-42 will need to be updated to use the new libraries.
- Exploit developed for RT-42 enabled a user to trigger the exploit through a simple text interface on the local machine; this triggering option will be impossible for the flight demonstration. Instead the trigger will be initiated either by the aircraft entering a specific geographic region or through a remote interface provided over the 2.4 GHz channel.
- Deep packet inspection functionality that is currently utilized in the CloudShield for RT-42 will be recreated using the Raspberry Pi platforms and the communications SDK to make the Sentinel flight-ready.
- Additional software development and testing will be needed to ensure the application is robust to potential failures.
- Exploit will need to be configured such that it is approved safe for a live flight demonstration.
- Exploit developed for RT-42 triggered the masking exploit on the ground using one of the user defined payload streams available on the Piccolo autopilot.
- Exploit will need to be re-written to utilize the TPU as discussed in section 4.1. To ensure that control over the aircraft is maintained at all times, the masking attack will be able to be started and stopped by a user located on the ground. This channel will be independent of the remote signal sent by the airborne exploit as well as capable of overriding it if needed.

4.2.1.4 Hardware Security against Design and Manufacturing Attacks

The Sentinel requires that the systems that interface with it standardize their interface protocols to an IP-based protocol for Sentinel monitoring, detection and potential restoration activities. To protect the Sentinel functionality we must protect the data protocol conversion process. In this case, the conversion is from RS-232, the main communications protocol for the Piccolo II autopilot, to the TCP/IP protocol for the Sentinel. In addition, we must have the same protections for TCP/IP protocol communications back from the Sentinel to RS-232 back to the autopilot. This is required for restoration command for other Sentinel protections.

Two of the hardware design attacks to the Sentinel's data conversion protocol will be implemented in the flight demonstration. These are the denial-of-service attack and the data spoofing attack described in section 3.2.4. The attack will be performed by a hardware Trojan embedded in one of the three soft-cores located on the FGPA platform on board the SiCore UAV SHIELD card. The hardware Trojan will monitor the RS-232 data flows. When the triggering pattern is recognized the hardware Trojan will initiate the attack. The attack will be demonstrated via two separate triggering mechanisms: (1) by predetermined GPS coordinates and (2) by injecting the appropriate signal from the ground station.

The three soft-cores will be implemented on the FPGA and will provide the main platform for providing protection for the data conversions on the Sentinel platform. We will implement diverse redundancy and verifiable voting security design patterns as protection mechanism for the data conversion processes for the Sentinel. The three soft-cores will also provide the platform for TMR methods to protect the data streams and conversion processes. A voter will be designed and developed to ensure that only protected data streams are sent in and out of the Sentinel.

4.2.2 Development of the UAV Sentinel

4.2.2.1 UAV SHIELD

The processor for the UAV SHIELD is a FPGA running three soft-core processors, which run applications developed by the UVa and the GTRI. These soft-core processors communicate with other hardware on the UAV through four Ethernet ports and eight RS-232 serial ports. It also has an SD card used for securely storing data. The proposed UAV Shield architecture is presented in Figure 38.

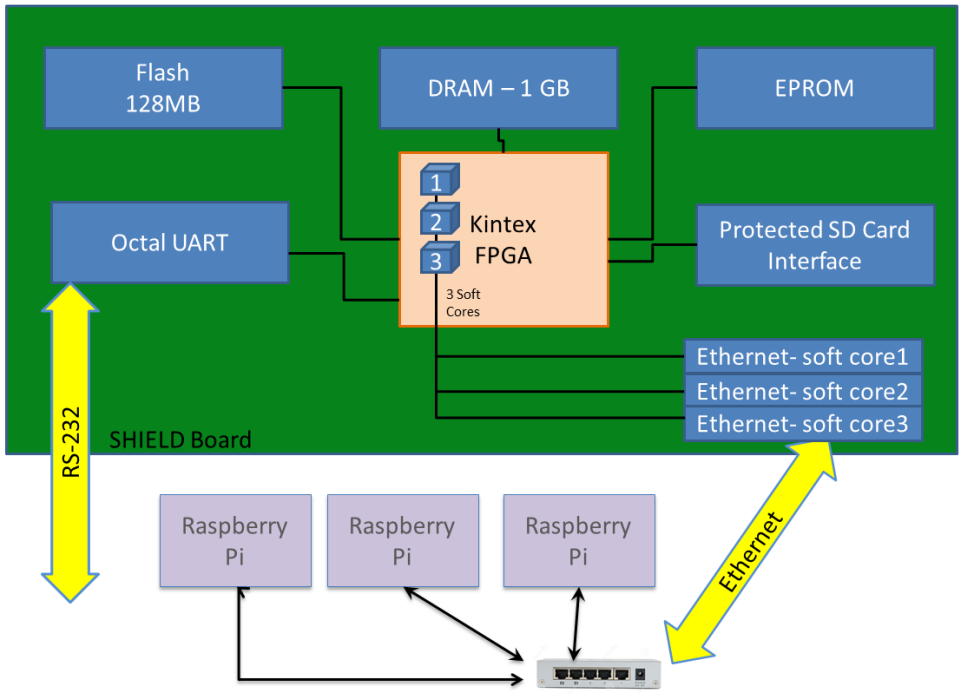


Figure 38. Proposed UAV SHIELD architectural block diagram.

4.2.2.1.1 NetFPGA-7

Many modifications were made to the design of the SHIELD Coprocessor to produce the design of the UAV SHIELD. The hardware of the UAV SHIELD is similar Computer Measurement Lab's (CML) and Sicore's NetFPGA-7 card (Figure 39). NetFPGA-7's capabilities include hardware mediation (SATA, Ethernet, Memory), which is controlled by the OODA Real-time Controller. This card will be used as a prototyping platform while the final version of the SHIELD Card for Sentinel (and for this project) is being fabricated.

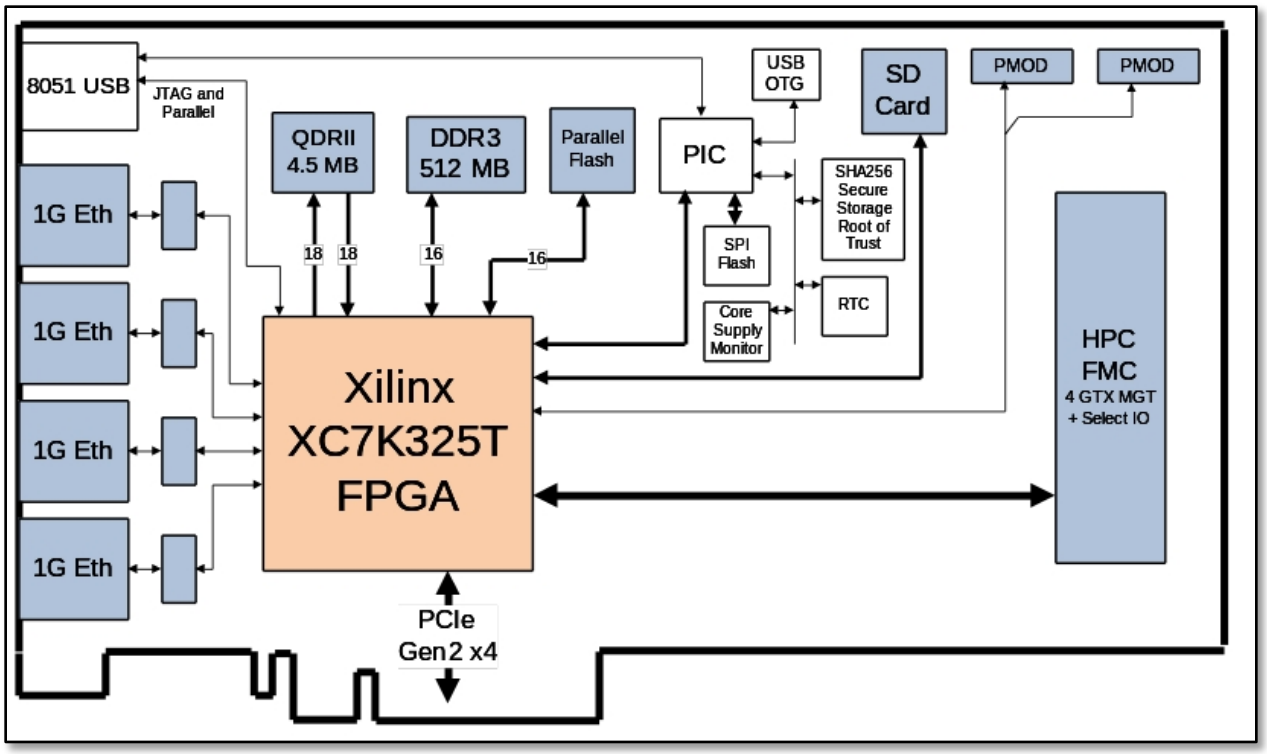


Figure 39. NetFPGA-7 architectural block diagram.

4.2.2.1.2 Modifications to SHIELD Card for the Sentinel

- Change FPGA to the Kintex 7 FPGA

  An FPGA needed to be selected with the capability of running three soft-core processors and establishing the integrity of the bitstream loaded to it. The Kintex 7 FPGA was chosen to specify these needs. It contains more than enough resources to instantiate the three soft-core processors and the interfaces to the Ethernet and RS-232 ports.

  For bitstream protection, at initialization, a cryptographic key associated with the bitstream is written to non-volatile tamper-resistant memory on the FPGA. Once written, a new key cannot be written to it. The compiler on the development machine encrypts the bitstream, which is then programmed to EEPROM on the UAV SHIELD. During power-up the bitstream from the EEPROM is written to the FPGA and decrypted. This process authenticates the bitstream and establishes its integrity. A bitstream written by an attacker and loaded to the EEPROM would fail to decrypt.

- 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

4.2.2.1.3 Potential SHIELD Security Features to be Selected for Final Implementation

This printed circuit board (PCB) is designed to support the development and demonstration of countermeasures against COTS supply chain corruption. These countermeasures are intended to defend against potential design logic, configuration bitstream and hardware mask exploits. Insider attacks within the supply chain are of particular concern, so multiple layers of defense are employed to help thwart potentially malicious insider activity that may occur at various links in the chain. Defensive capabilities are provided at the design logic, device configuration, and device mask levels. Diverse on-board subsystems help support these capabilities.

Core supply monitoring and multi-gigabit serial I/O is provided to help support individual device qualification against intentional mask corruption. This is made feasible via a combination of low frequency supply measurements and high-speed on-chip timing measurements. Design configuration bitstreams, although well-defended by manufacturer encryption methods, are still vulnerable to potential external key discovery, insider key theft and spoofing. Support for both vendor-independent end-to-end bitstream authentication and field installation authentication is provided to erect additional barriers against such attacks. In addition, the design logic itself can be corrupted even though bitstream encryption and programmable array electronics are intact. This attack vector is countered via the use of design-independent implementation verification to detect either static or dynamic corruption of the intended design logic.

These countermeasures combine to provide effective integrated defenses against hardware attacks at low incremental cost to the PCB design.

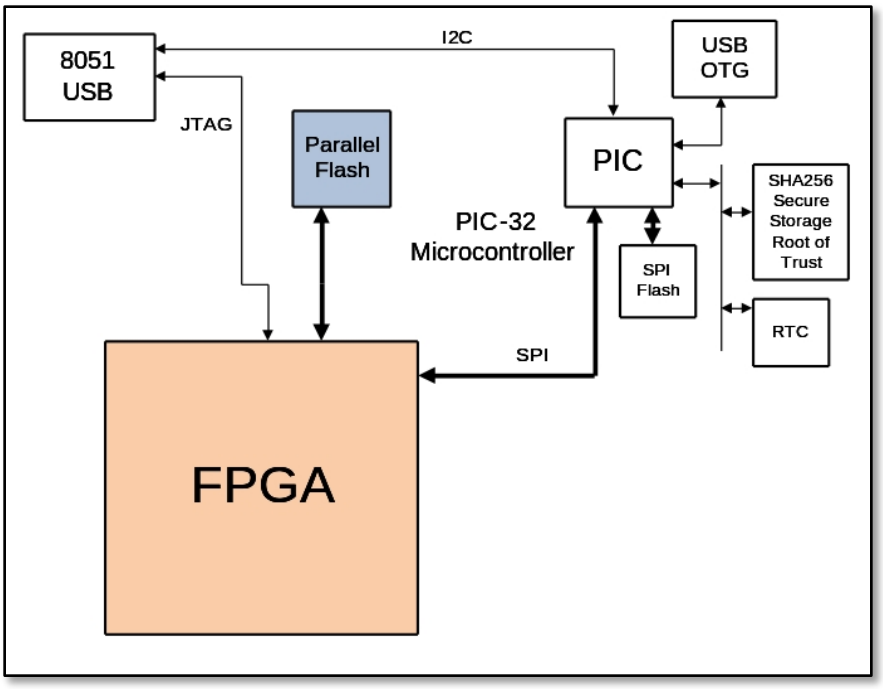


Figure 40. UAV SHIELD block diagram for the the FPGA with supporting cryptographic HW.

The OODA Real-time controller is a software package that implements the OODA Loop (Figure 41). The OODA Loop is a concept developed by military strategist and USAF Colonel John Boyd, that describes the method that individuals and organizations process and respond to events. Entities that can process the cycle quickly and intelligently can gain an advantage over their opponents. The loop consists of four major elements: Observe, Orient, Decide, and Act.

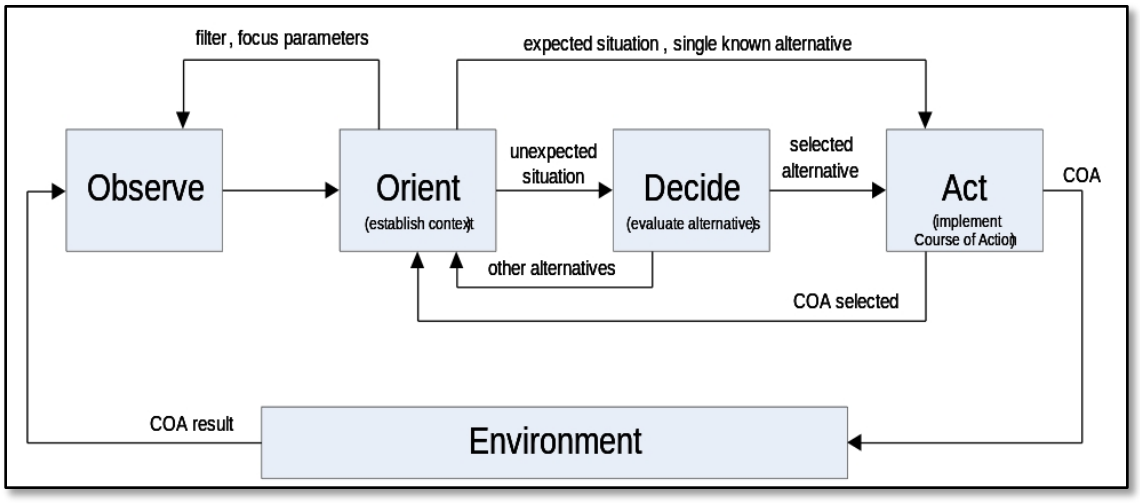


Figure 41. OODA Real-time controller that implements the OODA loop on the SHIELD Coprocessor.

The OODA Real-time Controller is a multi-threaded application that implements the observe, orient, decide, and act elements of the OODA loop in their own threads. It interfaces with various agent applications that gather observations about the environment and execute actions as directed by the OODA Real-time Controller. In the NetFPGA-7, these agents control the various hardware mediation capabilities.

The NetFPGA-7 card has hardware mediation capabilities (Figure 42). There are several benefits of hardware mediation:

- Establishing a defensive base independent of the host system.
- Monitoring capabilities to detect anomalous behavior.
- Implementing defenses without using host system resources.
- Deploying countermeasures at a hardware level 9.

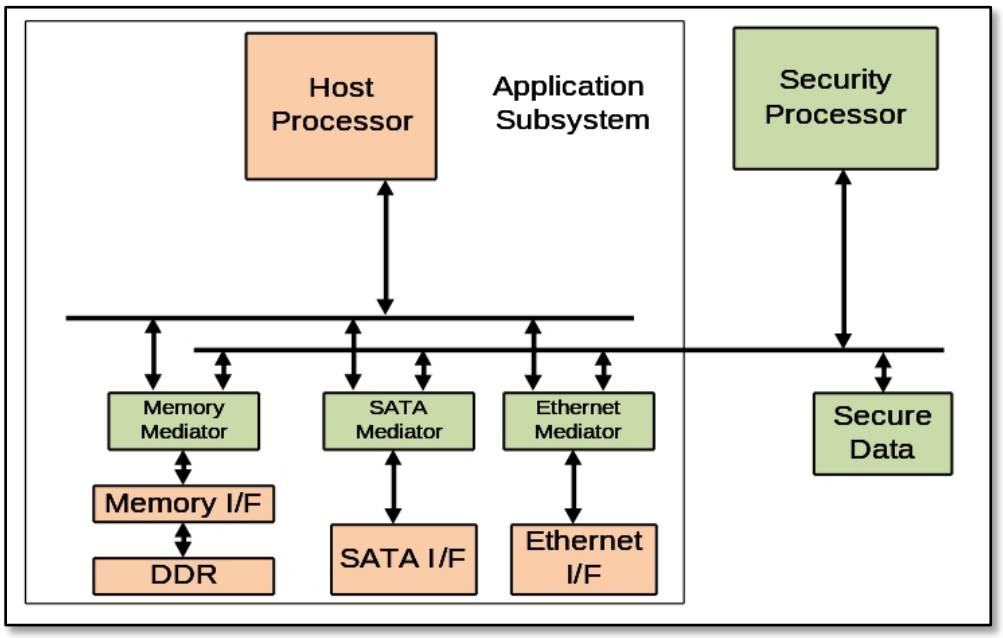


Figure 42. Block diagram of the NetFPGA-7 mediators.

The Ethernet Mediator is used for network intrusion detection. It captures ingoing and outgoing raw Ethernet packets, analyzes them, and forwards them to their destinations, during normal operation. It possesses several capabilities:

- Network packet monitoring and logging capabilities.
- Firewall capabilities that protect a specific system from external threats and detect malicious behavior on the host system.
- Synthetic packet injection.
- Packet redirection, through modifying the IP and MAC addresses.

The SATA Mediator protects the host system's hard drive. It captures all packets traversing between the host system's host bust adapter (HBA) and hard disk drive (HDD). The host system's HDD can also be accessed by firmware running on the NetFPGA-7 card without the host system's knowledge. The SATA Mediator provides several security capabilities:

- Hardware Write-Protection of partitions and files.
- File corruption detection.
- Autonomic restoration of corrupted files.

The memory mediator provides the user with the ability to covertly access the host memory. This capability allows the user to transparently and dynamically respond to malware. The memory mediator can observe and modify the host system's memory. Additionally, the mediator does not interfere with the host processor's access to memory. The host processor does not have any visibility into the memory mediator's actions. This enables cyber security systems to detect and repair subversion in near real-time. Detection and Mitigation development on new platform

4.2.2.1.4 Gimbal Attack Detection, Mitigation, and Restoration

The detection and mitigation of the gimbal attacks will follow the methods outlined in section 3.3.4. If the gimbal receives retract or angle commands that seem inappropriate in the context of the mission an alert will be issued to the operator and cyber-security office.

4.2.2.1.5 Parameter-Based System Attack Detection, Mitigation, and Restoration

The following modifications are required to deploy the System-Aware cyber security protections discussed in section 3.3.2 to defend against the parameter-based attack:

- CloudShield was chosen as the Sentinel platform for prototyping under RT-42. These protection mechanisms will need to be developed for the UAV SHIELD. As seen in Figure 37, three Raspberry Pi SBCs will form the platforms for hosting System-Aware algorithms for the UAV SHIELD Sentinel. Software developed for the CloudShield will need to be ported to this new environment:
    - Configuration hopping capabilities of the CloudShield prototype.
    - Cyber commander communications channel will be changed from an 802.11 wireless network to utilizing the user defined payload stream of the Piccolo autopilot.
- The airborne Sentinel will include the *Secure Voting* System-Aware design pattern. As this pattern was not part of the prototype developed under RT-42, additional development will be needed.
- Integration work to host the *parameter-based, GPS,* and *Gimbal* detection and restoration designs on the same platform will be required.
- Additional software development and testing will be needed to ensure the Sentinel security functions are robust to potential failures.
- Inclusion of *Diverse Redundancy* System-Aware design pattern through the implementation diverse (i.e., at least two) different implementations of the protection algorithms for the parameter-based-attack.

4.2.2.1.6 Hardware Security against Design and Manufacturing Attacks

As stated in section 3.3.5, we propose to apply TMR to the soft-core-based protocol conversion of data coming in from the RS-232 based communications from the Piccolo to the TCP/IP protocol for the Sentinel. This conversion process is a potential vulnerability that needs to be protected to ensure the Sentinel is able to fulfill its security role. The protocol converter between RS-232 and TCP/IP will be implemented on the UAV SHIELD board provided by SiCore. A block diagram of the UAV SHIELD board is shown in Figure 43.

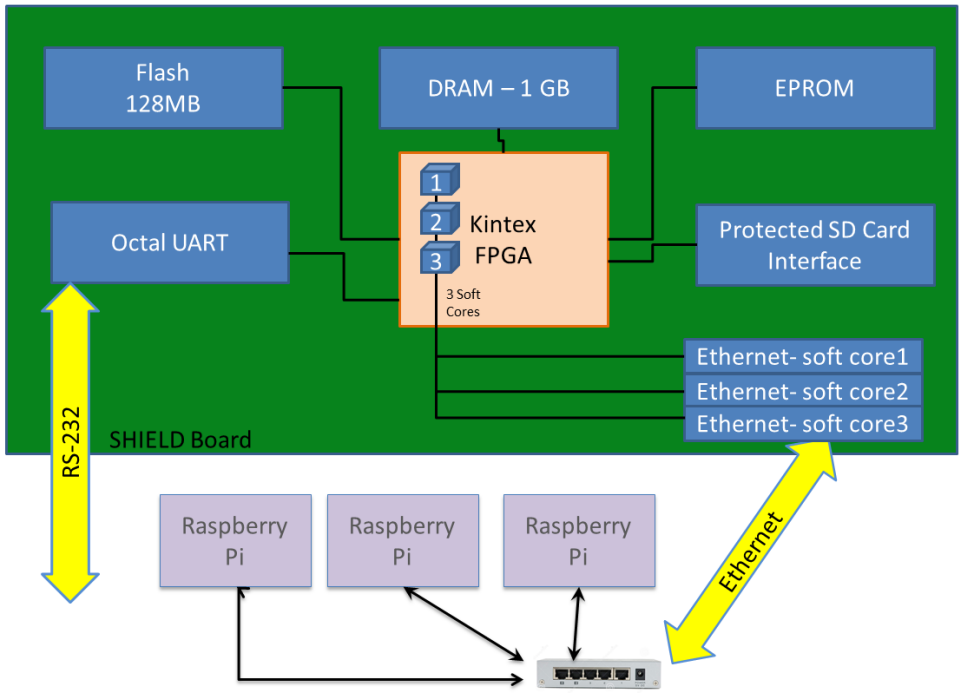


Figure 43. Block Diagram of the UAV SHIELD board.

The SHIELD board contains a user FPGA (Xilinx Zynq-7000 series). Three soft-cores and their peripherals will be implemented using the programmable logic in the FPGA in order to add the redundancy required for security considerations. Each processor will have its own peripherals such as UART interface, Ethernet interface, etc. Therefore TMR is applied to protect both the processors and the interfaces.

A majority voter will be implemented on the FPGA and will be used to generate a single output, masking the incorrect data stream of the corrupted core. Since the voter is not protected by TMR it could be vulnerable to attacks. To ensure the security of the voter, it must be developed by a trusted designer and thoroughly inspected. Since a voter's logic is straightforward only light inspection effort will be required. After the design process, the voter will be protected by the encryption and authentication of FPGA bitstream.

Protocol conversion will be done bi-directionally. In the RS-232-to-Ethernet (TCP/IP) direction a RS-232 data stream comes in on the UART interface and gets triplicated and sent to the three soft-cores. When the conversion is done the voter takes the data from the three cores and determines the result. The output of the voter is then sent to the Ethernet interface, which wraps the data into Ethernet packets. The Ethernet packets will be sent to three Raspberry Pi's within the Sentinel through a network switch. Therefore, the Ethernet interface will send each packet in the data stream to three different IP addresses.

In the Ethernet-to-RS-232 direction the data flow is the opposite of the aforementioned process. Again, a voter will be used to generate a single data stream before sending it back put to the UART interface.

Protocol conversion will be performed by software which runs on the soft-cores on the FPGA. The three soft-cores will share 1 GB DRAM on the UAV SHIELD board. The EEPROM will be used for storing the FPGA configuration bitstream. The software (a bare-metal application) for protocol conversion will be stored in the flash memory on board the SHIELD board.

The UAV SHIELD board contains a user FPGA. For the purpose of prototyping, we propose to use LEON3 soft-core processor. LEON3 is an open source synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The model is highly configurable. This allows the implementation of cores with different configurations, which is desirable for the purpose of this project. In addition, LEON3 is device-independent, which makes the design portable to other devices in the future. For prototyping purposes, we will also make use of the NetFPGA7 card (described in section 4.2.2.1.1) from SiCore that is similar in design to the final UAV SHIELD configuration while that final card configuration is being fabricated.

4.2.2.1.6.1 Detection of Attacks in the Post-design Phase

In order to detect more general attacks in the supply chain, such as reverse engineering, tampering with the design and replacement of FPGA devices, encryption and authentication of the FPGA bitstream is required.

The Xilinx FPGA employs AES and RSA for the purpose of bitstream encryption and authentication, respectively. These features can be used to detect and prevent supply chain attacks. Therefore the detection of general attacks in the post-design phase supply chain can be done by this built-in feature.

The triplicated soft-cores, their peripherals, as well as extra circuits for the purpose of attacking, detection, and restoration will be implemented on the FPGA. The three soft-cores will share 1 GB DRAM on the SHIELD board. The EEPROM will be used for storing the FPGA bitstream. The bare-metal application for protocol conversion will be stored in the Flash memory.

A long-term goal is to investigate the feasibility of using ARM-based cores from different vendors. One of the advantages of the ARM architecture is that it is widely used and therefore many options are available.

We also propose as a long-term goal to prototype for discrete ICs, and to develop IC design rules in terms of hardware security, as well as board-level design patterns that can be generalized to any digital IC and chip sets.

In addition, we propose to apply PUFs to the hardware. PUFs provide a way to generate a *fingerprint* of a circuit and therefore can be used to authenticate a FPGA device itself.

Furthermore, recovery is required to distinguish between a transient fault caused by an SEU and a persistent error caused by an attack. For example, if an SEU flips a bit in a FIFO, recovery is not necessary because the data is used only once. However, if an SEU flips a bit in a memory, then recovery is required because the same location could be read for multiple times. Both situations mentioned above should be distinguished from an attack, in which case the malfunctioning core will be kicked out of the system.

4.2.3 Design and Build Integrated System

The platform described in Figure 37 describes the new Sentinel platform. The on board, flight-ready version of the Sentinel is comprised of the new SiCore UAV Shield card, three Raspberry Pis, and a small network switch. In addition to the Sentinel hardware, there are two attack platforms (two Raspberry Pis) that will serve as snoopers and malware injectors into the autopilot and gimbal systems.

The software components that have been developed separately for each of the attack scenarios will be integrated on to the hardware platform described in Figure 37. In addition to the attack platforms, Phase 2 efforts will also focus on integrating the Sentinel monitoring, detection and restoration methodologies that have been designed and tested separately in Phase 1 into an integrated Sentinel platform on board the flight-ready Sentinel hardware. Software development and integration activities will run through June 2014.

4.2.3.1 Build onboard package for Sentinel and Interfaces

The components of the onboard Sentinel and attack platforms will have a supporting infrastructure on the airframe. This will include a casing for the hardware components and the associated wiring harnesses needed to support the interfaces for the Sentinel and the connections for the network and attack platforms. The build of the physical hardware environment with software on board will be completed by July, 2014

4.2.4 Ground Testing

Extensive ground testing will be conducted prior to the flight demonstration. Initial testing will be conducted with individual components, subassemblies, and the complete system in a bench-top environment. Afterwards the complete system will be installed in the aircraft for further ground testing. The following subsections describe the ground test activity. Ground tests will be completed by the middle of September, 2014.

4.2.4.1 Test integrated system in a HiL/bench-top environment

Initially the Sentinel components will be tested using the HiL emulators at the UVa and GTRI. After the components have been verified as functioning properly, they will be integrated to form the complete

Sentinel system. This system will then be tested in the GTRI HiL emulator which best replicates the GAUSS UAV. During the HiL emulator testing, all of the hardware and software components for the attacks, detections, and mitigation will be verified to be functioning properly.

4.2.4.2 Move solution to airframe for on-ground testing

Once the HiL emulator testing has verified that the system is working properly on the bench-top, the system will be installed in the GTRI GAUSS UAV. This will allow testing of the system with the actual onboard power supplies and radio frequency (RF) modems in conjunction with the GTRI HiL emulator. Testing will be conducted with the engine running to make sure that the vibration and electromagnetic environments are acceptable. The test plans for the demonstration flights will be executed during the emulator testing as a rehearsal for the actual flights.

4.2.5 Flight Testing

Two flight demonstrations are planned following the completion of the ground tests using the HiL emulator. The flights will be conducted at the Early Co. airport in southwest Georgia. GTRI has a Certificate of Authorization (COA) from the FAA to operate the GAUSS UAV at this general aviation airport. The flights will take place within the boundaries of the COA shown in Figure 44. Flight testing will run from the October through the middle of November, 2014.

Figure 44. COA Boundaries for Early Co., GA.

4.3 Required Activities, Distribution of Effort, Deliverables, Costs and Schedules

A Gantt chart of the proposed schedule for Phase 2 is presented in **Table 17** in the Appendix.

4.4 On-going Evaluative Questions and Early Outcomes

As discussed in section 3.4, there are several research questions related to the implementation of a super secure Sentinel for UAS that we hope to address in both Phase 1 and Phase 2 of this project. As we progress through the implementation of the Sentinel for the UAV flight system, we will continuously evaluate the practicality of each of the proposed measurements and their associated analysis algorithms for completion within this phase of the project and present rational for cases where it proves to be impractical while including suggestions that will serve as the basis for future implementation strategies.

What are potential attacks?

As outlined in section 3.1.3.3, our team identified and selected four exploits to protect against:

1. Waypoint, parameter attack.
2. GPS navigation attack.
3. Gimbal / Camera attack.
4. Attack against the data conversion interfaces.

What are the available data measurements from the system to be monitored?

The documentation for the Piccolo autopilot provides significant detail about the physical layout of the autopilot and the details of the structures of the streams of data that serve to control the system. These streams include telemetry data structures that report the orientation of the aircraft and its control surfaces, payload data that reports mission data, gimbal stream that reports the status of the aircraft's TASE camera gimbal system, and general status packets which constantly update the status of the various sub-systems controlled through the Piccolo autopilot. This off-the-shelf documentation is available to customers of who have purchased the Piccolo autopilot.

As we look into what data needs to be collected for use in evaluating the performance of the Sentinel, there are additional data that needs to be collected and additional analytical processes that need to be addressed. For example, considering the attack on the GPS data affecting the imagery metadata, we must not only track the existing autopilot GPS stream, but also analyze the data coming from the other diverse, redundant GPS systems to be able to accurately run detection algorithms, and to classify deviations as either cyber-attacks or system failures with desired likelihood of detection and acceptable false alarm rates. The readings from that system must be standardized into common timing and common formatting for comparisons within the Sentinel architecture and time series of data will need to be recorded for use in evaluation of Sentinel performance. In addition, to evaluate the collateral impacts of the Sentinel on the systems that are being monitored, we will need to collect data from tests

that have the Sentinel in place, and compare the results with test cases where the Sentinel is not engaged to help gauge the impact of running the protection algorithms.

As part of the next phase, we also will need to need to collect data that will be generated by the Sentinel, so as to produce data sets that that will allow us to evaluate the performance of different design patterns within the Sentinel design, and analyze that data to look closely at the ability to detect the attacks as they occur, and the performance of the system after a restoration to determine if we have returned to normal systems operation. This will be vital in scenarios where attacks continuously repeat themselves and when continuous corrective actions may be necessary to keeps the system functioning.

Ultimately, we would like to evaluate the feasibility of recording the entire data stream from the autopilot, the gimbal system and the Sentinel in order to enable our ability to replay the entire flight and use the data for forensic analysis. Where that data is collected and the realities of the physical limitations in the on-board system may dictate how much data we can collect and where the data will be stored.

The project team will formalize the data plan for evaluative purposes and that plan will provide the basis for the data logging mechanisms inserted into the hardware platform and also for the algorithms that will be used to evaluate the performance of the Sentinel protection and their effects on the monitored subsystems.

What should be measured to protect against potential cyber-attacks

Data measurements needed to protect the UAV against the identified exploits are outlined in section 3.4**.**

In terms of measuring the ability to collect data needed to evaluate the ability of the Sentinel to protect the system function under attack, Phase 2 of this project will help to identify the actual data measures that are needed to evaluate Sentinel performance. This will include data that is recorded on the ground and in the air and will also reflect data that is collected from both the system itself, in this case the autopilot, gimbal and ground components, and the Sentinel architecture itself.

Can we standardize the data that is provided by the various interfaces?

As part of the investigation of the use of Sentinel protections, the UVA/GTRI team has addressed the need to standardize the data traffic streams for analysis. For this system, RS-232 serial communications were converted to IP (Internet Protocol) based data packets which can be used for monitoring, detection, and for system restoration. As we look into protecting other systems, each will have its own set of data protocols that will need to be put into a standardize format that the Sentinel technology can use. The proliferation of the IP protocol and the large number of interfaces that allow for easy conversion to the IP protocol make it a logical candidate for a design standard. The implications to system latencies, timing, and other collateral effects will need to be investigated. Bench testing has shown that this serial to IP conversion process has been stable and reliable. However, as we move into a

physical implementation, the latencies associated with the conversion process will need to be evaluated for collateral system impacts.

The conversion from the serial data stream to IP will be protected by a triple redundant validation scheme that will aim to ensure that the data being input to the Sentinel can be trusted. However, this protection potentially introduces additional overhead and latencies that will need to be evaluated and data will need to be produced to make that evaluation possible.

What is the frequency of the data measurements that we need to extract to adequately detect system state changes that might indicate a cyber-attack?

In general, the frequency of data measurements needed to defend against a potential cyber-attack is in human terms; i.e., seconds and minutes. For example, the system transmits status message containing detailed information about the state of the aircraft approximately every 6 seconds, and this status message is the most prevalent data packet in the serial data streams we observed. As outlined in section 3.4, this rate of data measurement is perfectly adequate for detecting, defending, and restoring from several potential cyber-attacks. However, the time sensitivity of system functions may vary across the different sub-systems and each may have different timing issues that need to be addressed to monitor them, and also to utilize the interfaces as mechanisms for restoration when they are under attack. In addition, we must be able to determine the sensitivity of the system in terms of false detections (or false alarms) versus increased security.

For each of the attack scenarios, there will be a difference between when the initiation of an attack occurs, the detection of the attack, and the time it takes for a restorative action. Those time differences will ultimately drive us to what state we can or want to put the system back to and how we manage that restoration while continuing normal system operation. For example, to protect the gimbal system we are investigating the use of mission context for the evaluation and classification of a system change. If the monitor sees a change in system commands, we must evaluate whether the context provided is adequate to allow the change to happen or if the system should block that change. Ultimately, the sensitivity of the change to the context will affect the number of false detections. This also raises the need to have experimentation to help decide issues such as when to provide an alert, when to allow a change because it is easily recoverable, when to block the change, etc. These kind of issues also allow the human (i.e. the operator or the cyber officer) to have some input into that decision making process.

What are the methods needed for assuring the integrity of an operation?

The methods used for realizing System-Aware security are outlined in section 3.1.

What is the complexity of the algorithms used for securing the system to be protected?

As outlined in section 3.3.1, the algorithms used for protecting the UAV against the selected exploits have been relatively small; i.e., hundreds of lines of code. However, the complexity of these algorithms may change as the project transitions from HiL emulation to flight-ready hardware. In addition, changes may be necessary as these algorithms are implemented in the UAV SHIELD card used to implement the

Sentinel. Thus, part of the effort for Phase 2 will be to see how the complexity of these algorithms change based upon the constraints imposed by a UAV.

How should the Sentinel respond once an attack has been detected?

As discussed in section 3.3, as the project has progressed, it was decided to introduce a specially designated cyber security officer to facilitate decisions about how best to respond during a cyber-attack. Phase 1 has focused on identifying the need for and the role of the cyber security officer in ensuring that missions are completed successfully; e.g., minimizing the workload on the aircraft's pilot, as well as the need for an individual with extensive knowledge of cyber-attacks. A parallel effort is being pursued to focus on the information that should be provided to the cyber security officer, as well as how this new individual should be integrated into the existing workflow for conducting ISR missions.

5 Appendix

5.1 Supporting Calculations for GPS System Attack

Proposed Mass Function for FOD (3 components)

Event 1 and Even 8:

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

Event 2 and Event 7:

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

Event 3 and Event 6:

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

Event 4 and Event 5:

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

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

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

5.2 Code Examples for GPS

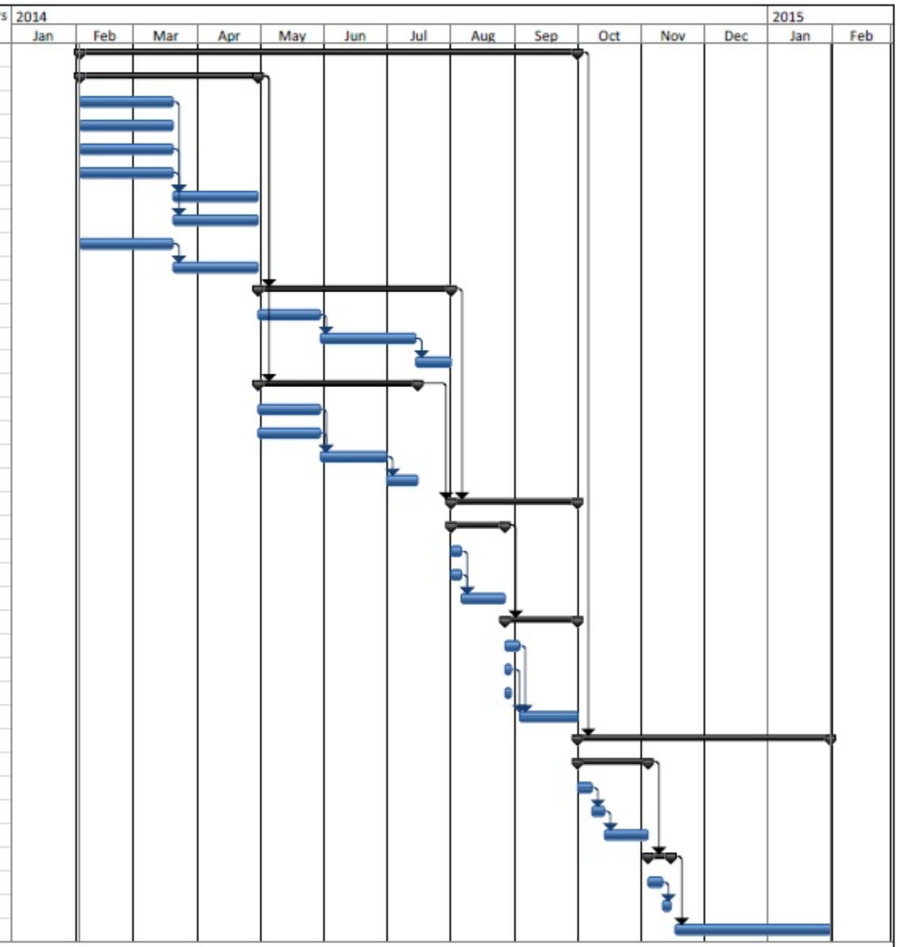
Below is the C/C++ source code for :

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159265

using namespace std;

double deg2rad(double deg)
{
```

```
   return deg*(PI/180);

}

double getDistance(double lat1, double lon1, double lat2, double lon2)
{
   double R = 6373; //Radius of the earth in km
   double dLat = deg2rad(lat2-lat1); //deg2rad see below
   double dLon = deg2rad(lon2-lon1);
   double a = sin(dLat/2)*sin(dLat/2)+
      cos(deg2rad(lat1))*cos(deg2rad(lat2))*
      sin(dLon/2)*sin(dLon/2);
   double c = 2*atan2(sqrt(a),sqrt(1-a));
   double d = R*c; //Distance in km

   return d;
}
```

Project Plan and Timeline for Phase 2 Activities

ID	Outline Numbe	Task Name	Duration	Start	Finish	Predecessors
1	**1**	**Phase II - System Integration & Ground Test**	**172 days**	**Mon 2/3/14**	**Tue 9/30/14**	
2	**1.1**	**Design & Build Sentinel System**	**62 days**	**Mon 2/3/14**	**Tue 4/29/14**	
3	1.1.1	Hardware design	33 days	Mon 2/3/14	Wed 3/19/14	
4	1.1.2	Software design	33 days	Mon 2/3/14	Wed 3/19/14	
5	1.1.3	Develop attack code	33 days	Mon 2/3/14	Wed 3/19/14	
6	1.1.4	Develop triggering mechanisms	33 days	Mon 2/3/14	Wed 3/19/14	
7	1.1.5	Develop attack platforms	29 days	Thu 3/20/14	Tue 4/29/14	5,6
8	1.1.6	Build Sentinel platform	29 days	Thu 3/20/14	Tue 4/29/14	3
9	1.1.7	Identify data collection needs	33 days	Mon 2/3/14	Wed 3/19/14	
10	1.1.8	Design data collection and logging techniques	29 days	Thu 3/20/14	Tue 4/29/14	9
11	**1.2**	**Bench-top Prototype Sentinel Evaluation**	**67 days**	**Wed 4/30/14**	**Thu 7/31/14**	**2**
12	1.2.1	Develop test scripts	22 days	Wed 4/30/14	Thu 5/29/14	
13	1.2.2	Conduct bench-top testing	32 days	Fri 5/30/14	Mon 7/14/14	12
14	1.2.3	Revise data collection and tests scripts	13 days	Tue 7/15/14	Thu 7/31/14	13
15	**1.3**	**Build onboard package for Sentinel and Interfaces**	**55 days**	**Wed 4/30/14**	**Tue 7/15/14**	**2**
16	1.3.1	Develop schematics and wiring diagrams	22 days	Wed 4/30/14	Thu 5/29/14	
17	1.3.2	Design installation fixtures	22 days	Wed 4/30/14	Thu 5/29/14	
18	1.3.3	Fabricate fixtures and wiring harnesses	22 days	Fri 5/30/14	Mon 6/30/14	16,17
19	1.3.4	Perform weight & balance analysis	11 days	Tue 7/1/14	Tue 7/15/14	18
20	**1.4**	**Ground Testing**	**43 days**	**Fri 8/1/14**	**Tue 9/30/14**	**15,11**
21	**1.4.1**	**Test integrated system in the GTRI HiL**	**18 days**	**Fri 8/1/14**	**Tue 8/26/14**	
22	1.4.1.1	Preliminary flight test plans	3 days	Fri 8/1/14	Tue 8/5/14	
23	1.4.1.2	Preliminary data collection plan	3 days	Fri 8/1/14	Tue 8/5/14	
24	1.4.1.3	Conduct HiL bench-top testing	15 days	Wed 8/6/14	Tue 8/26/14	22,23
25	**1.4.2**	**Move solution to airframe for on-ground testing**	**25 days**	**Wed 8/27/14**	**Tue 9/30/14**	**21**
26	1.4.2.1	Install system components in aircraft	5 days	Wed 8/27/14	Tue 9/2/14	
27	1.4.2.2	Revise flight test plan	3 days	Wed 8/27/14	Fri 8/29/14	
28	1.4.2.3	Revise data collection plan	3 days	Wed 8/27/14	Fri 8/29/14	
29	1.4.2.4	Conduct HiL testing with aircraft	20 days	Wed 9/3/14	Tue 9/30/14	27,26
30	**2**	**Phase III - Flight Test**	**88 days**	**Wed 10/1/14**	**Fri 1/30/15**	**1**
31	**2.1**	**First flight test**	**24 days**	**Wed 10/1/14**	**Mon 11/3/14**	
32	2.1.1	Vehicle prep	5 days	Wed 10/1/14	Tue 10/7/14	
33	2.1.2	Early Co. Demo 1	4 days	Wed 10/8/14	Mon 10/13/14	32
34	2.1.3	Post-flight review	15 days	Tue 10/14/14	Mon 11/3/14	33
35	**2.2**	**Second flight test**	**9 days**	**Tue 11/4/14**	**Fri 11/14/14**	**31**
36	2.2.1	Vehicle prep	5 days	Tue 11/4/14	Mon 11/10/14	
37	2.2.2	Early Co. Demo 2	4 days	Tue 11/11/14	Fri 11/14/14	36
38	2.3	Post Flight-Test Analysis	55 days	Mon 11/17/14	Fri 1/30/15	35



Project: Flight test budget_v2
Date: Thu 11/14/13

Task	Project Summary	Inactive Milestone	Manual Summary Rollup	Deadline
Split	External Tasks	Inactive Summary	Manual Summary	Progress
Milestone	External Milestone	Manual Task	Start-only	
Summary	Inactive Task	Duration-only	Finish-only	

Page 1

Table 17. Proposed schedule for Phase 2.