NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CYBER SECURITY TESTING OF THE ROBOT OPERATING SYSTEM IN UNMANNED AERIAL SYSTEMS

by

Sergio Sandoval

September 2018

Thesis Advisor:	Preetha Thulasiraman
Second Reader:	Murali Tummala

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE CYBER SECURITY TESTING OF THE ROBOT OPERATING SYSTEM IN UNMANNED AERIAL SYSTEMS			**5. FUNDING NUMBERS** R4M3G
6. AUTHOR(S) Sergio Sandoval			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) CRUSER/ONR			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Unmanned systems have gained in prominence as platforms from which to conduct military operations. The Robot Operating System (ROS) is a widely adopted standard robotic middleware; however, its preliminary design is devoid of any network security features. Military grade unmanned systems must be guarded against network threats. ROS 2.0 is built upon the Data Distribution Service standard and is designed to provide solutions to identified ROS 1.0 security vulnerabilities by incorporating authentication, encryption, and process profile features. The Department of Defense is looking to use ROS 2.0 for its military-centric robotics platform. Through our work, we demonstrated that ROS 2.0 can serve as a functional platform for use in military grade unmanned systems. We tested the viability of ROS 2.0 to safeguard communications between an unmanned aerial swarm and a ground control station against rogue node and message-spoofing attacks. Our experiments employ the PX4 Multi Vehicle Simulation swarming three iris-quadcopter aerial drones within a Gazebo 9 simulation environment, utilizing QGroundControl as our ground control station. Drones were targeted individually to ascertain the effectiveness of our attack vectors under specific conditions. We demonstrated the effectiveness of ROS 2.0 in mitigating the chosen attack vectors but observed a measurable operational delay within our simulations.

14. SUBJECT TERMS authentication, authorization, encryption, ROS, unmanned aerial vehicle, swarm, security			**15. NUMBER OF PAGES** 55
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

CYBER SECURITY TESTING OF THE ROBOT OPERATING SYSTEM IN UNMANNED AERIAL SYSTEMS

Sergio Sandoval
Major, United States Marine Corps
BS, University of California - Los Angeles, 2006
MA, Webster University, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2018

Approved by: Preetha Thulasiraman
Advisor

Murali Tummala
Second Reader

Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned systems have gained in prominence as platforms from which to conduct military operations. The Robot Operating System (ROS) is a widely adopted standard robotic middleware; however, its preliminary design is devoid of any network security features. Military grade unmanned systems must be guarded against network threats. ROS 2.0 is built upon the Data Distribution Service standard and is designed to provide solutions to identified ROS 1.0 security vulnerabilities by incorporating authentication, encryption, and process profile features. The Department of Defense is looking to use ROS 2.0 for its military-centric robotics platform. Through our work, we demonstrated that ROS 2.0 can serve as a functional platform for use in military grade unmanned systems. We tested the viability of ROS 2.0 to safeguard communications between an unmanned aerial swarm and a ground control station against rogue node and message-spoofing attacks. Our experiments employ the PX4 Multi Vehicle Simulation swarming three iris-quadcopter aerial drones within a Gazebo 9 simulation environment, utilizing QGroundControl as our ground control station. Drones were targeted individually to ascertain the effectiveness of our attack vectors under specific conditions. We demonstrated the effectiveness of ROS 2.0 in mitigating the chosen attack vectors but observed a measurable operational delay within our simulations.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

UxS	Unmanned Systems
UAV	Unmanned Aerial Vehicle
ROS	Robot Operating System
OSRF	Open Source Robotics Foundation
ROS-I	Robot Operating System Industrial
ROS-M	Robot Operating System Military
DoD	Department of Defense
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
API	Application Programming Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XMLRPC	Extensible Markup Language Remote Procedure Call
IP	Internet Protocol
SROS	Secure ROS
OMG	Object Management Group
DDS	Data Distribution Service
RTPS	Real-Time Publish Subscribe
3DES	Triple-Data Encryption Standard
NIST	National Institute of Standards and Technology
AES	Advanced Encryption Standard
MAC	Message Authentication Code
TLS	Transport Layer Security
PKI	Pubic Key Infrastructure
TARDEC	Tank Automotive Research Development and Engineering Center
GCS	Ground Control System
MITM	Man-in-the-Middle
DoS	Denial-of-Service
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie-Hellman

CA	Certificate Authority
AES	Advanced Encryption Standard
AES-GCM	AES in Galois Counter Mode
AES-GMAC	AES Galois Message Authentication Code

ACKNOWLEDGMENTS

Many individuals have generously imparted their wisdom and devoted their time to make this endeavor successful. To my cohort in the Electrical and Computer Engineering program, you have been a source of strength. Thank you for your friendship and encouragement. To my instructors, each of you has imparted upon me the wisdom to succeed in not just my academic life, but in my personal world.

To the knowledgeable and talented folks at the Open Source Robotic Foundation, I could not have done this work without your assistance. In particular, I want to acknowledge Mikael Arguedas for opening my eyes to the possibilities.

To Professor Preetha Thulasiraman, my advisor, I cannot thank you enough for your keen insight and patience. You have been an incredible source of motivation guiding my efforts while ensuring that I stayed the course.

Above all, I would like to express my dearest appreciation to my exceptional spouse, Aisha, and my wonderful child, Sergio, for their admiration, curiosity, kindness and support. They are a marvel to behold.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. IMPORTANCE OF UNMANNED SYSTEMS

Unmanned systems (UxS) have been growing in prominence as platforms from which to conduct or support military operations. The U.S. military's use of these systems is changing the face of network centric warfare and altering the decision making process in combat operations. In the past several decades, the military's use of unmanned systems in all domains, including ground, air, surface, and subsurface, has increased dramatically. Military communities use UxS for the execution of offensive, reconnaissance, and surveillance missions. One of the goals of research and development in UxS is to ensure that they are network enabled and cyber hardened against adversarial interference.

Since 2016, the Naval Postgraduate School has been studying the operational impact of cyber threats on UxS, particularly unmanned aerial vehicles (UAVs) [1]. The vulnerabilities of the UAV architecture include threats against the communications link between a UAV and ground station as well as the communications between individual UAV assets. The communication link is only one of several inputs that are susceptible to cyber-attacks. One of the understudied areas of UAV security is the sensitivity of the Robot Operating System (ROS) to external threats.

ROS is a robust, open source, general-purpose platform that is used for robotics programming. Developed by the Open Source Robotics Foundation (OSRF), ROS is comprised of a set of software repositories and tools that assist in the development of robotics applications [8]. Variants of ROS have been developed in recent years, including ROS for industry applications (ROS-I) [2] and ROS for military applications (ROS-M) [3]. ROS-I has been influential in industrial automation systems like supervisory control and data acquisition systems (SCADA). Similarly, Department of Defense (DoD) organizations are seeking to build a military-centric ROS that provides an open, modular architecture with a library of military-unique components and a set of military-unique tools. The implementation of ROS-M is currently in its concept development phase [4].

B. THE UTILITY OF ROS

The ROS is a widely adopted standard robotic middle-ware. The ROS middleware sits on top of a host operating system providing a communications layer that supports the construction of functional computer clusters that are tailored for robotics applications [5]. Acting as a multi-server distributed computing network, the ROS structure allows software applications the ability to connect and communicate across server boundaries [6]. This aspect allows ROS-supported software applications to act as one single software system. The master node sits at the center of the ROS topography. This dedicated server is responsible for registering applications as well as their execution [6]. It also serves as a repository for parameters and for the logging of message traffic [6]. In addition to the ROS master node, a constellation of servers exists to balance out the system through the running of additional applications. These satellite servers connect to the master server through use of the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) via the local network.

Within this structure, ROS users implement independent systems that are referred to as nodes. A node can be used for explicit purposes with specific functionality, such as the execution of robot commands, the processing of transmitted message data, and as a means to receive and update sensor data [7]. Each of these nodes is inexorably tied to the ROS master. It is their connection to the ROS master that enables individual nodes to locate and communicate with one another [7]. Once this connection is made, they can communicate by passing messages. Nodes are further classified based on how they interact with prescribed topics within the ROS structure. Nodes that act as a receiver of information from a specific topic are said to be subscribed to that topic, while nodes that wish to send out information about a specific topic are referred to as publishers [7]. Nodes intercommunicate through an Application Programming Interface (API) via an Extensible Markup Language Remote Procedure Call (XMLRPC). XMLPRC is a remote procedure call protocol that uses XML encoding. Nodes can further intercommunicate through message and service data exchanges requiring the use of transport libraries such as ROSTCP and ROSUDP, which serialize communications through Internet Protocol (IP) sockets [5].

The original development of ROS focused on software attributes which robotics researchers coveted most. The relative simplicity and user-friendly aspects of ROS have made it a popular instrument for the robotics community; however, interest in ROS and its viability for application in industrial and military operations has demanded a modification to the system to enhance its security.

C. ROS 1.0

The very nature of ROS as an open source platform presents a great deal of concern for military operators. The system and its design and functions are available to anyone with access to the Internet. Moreover, the developers and maintainers of ROS provide a support apparatus on the GitHub webserver. A user can directly communicate with developers and with the ROS community to troubleshoot systems and enable the operating system's functionality. Beyond the GitHub site, there is comprehensive published literature that provides a user's guide to ROS [8].

ROS 1.0 was designed without any network or cyber security features mainly because it was designed for research purposes. The ROS 1.0 architecture does not have security features for communications between nodes. There exists no means of encryption nor authentication between nodes. This presents a problem as these networked nodes exist to facilitate the actions and motions of physical systems enabling the real-world actions of a robotics system. An adversary has a multitude of vectors by which to exploit a ROS 1.0 enabled unmanned system. The security vulnerabilities inherent in ROS 1.0 within the context of UAV swarm operation are discussed in Chapter III. In addition, mitigation techniques for these exploits are provided.

D. ROS 2.0

In recent years, ROS research has focused on the demand for network security features, namely identity authentication and authorization of resource permissions [5]. Academics and researchers have taken steps to allay security vulnerabilities in the ROS 1.0 platform through the application of varying security enhancements, which are layered on top of ROS 1.0. OSRF remedied the identified security limitations of ROS 1.0 by creating Secure ROS (SROS). SROS was designed to provide solutions to common security vulnerabilities by way of encryption, authentication, and process profile features [5]. SROS was incorporated within the ROS 1.0 platform. It does not exist as a standalone system that can function as its own version of the ROS API.

ROS 2.0 was developed with all of the security features that SROS introduced built into the system itself. ROS-M is based on ROS 2.0. The emphasis for ROS 2.0 is on the middleware, which is built atop the Object Management Group (OMG) consortium Data Distribution Service (DDS) standard. DDS is an open standard for developing real-time mission-critical distributed systems and incorporates the eProsima Fast Real Time Publish Subscribe (RTPS) protocol [9]. Fast RTPS provides publisher-subscriber communications over unreliable transports such as UDP [10]. DDS is used throughout government and industry, including extensive deployment throughout the DoD.

E. THESIS CONTRIBUTIONS

Through our work, we seek to demonstrate that ROS 2.0 can serve as a functional platform for use in military grade UxS. In this thesis, we focus on the viability of ROS 2.0 to safeguard communications between swarms and a ground control station. We test ROS 2.0’s ability to mitigate certain specific communications threats including message spoofing and rogue nodes. We use the underlying security processes available in DDS with the Fast RTPS model to execute our simulations. DDS enabled with the Fast RTPS protocol provides security through authentication, access control, and encryption.

Our experiments were executed in Gazebo, a simulation platform for robotic systems. We incorporated three quadrotor iris drones and a ground station controller, specifically QGroundControl, utilizing ROS 2.0. A bridge between ROS 2.0 Ardent and ROS 1.0 Kinetic was created to enable the security features inherent within ROS 2.0. Tests were first executed with the ROS 1.0 system alone, followed by the ROS 2.0-ROS 1.0 bridged system, and finally the bridged system with security features enabled. Drones were targeted individually to ascertain the effectiveness of our attack vectors.

While the solution provided in this thesis is specific to UAV swarms, the results can be applied generally to any unmanned system that incorporates the use of the ROS API.

F. THESIS ORGANIZATION

The remainder of this thesis is organized as follows. In Chapter II, the related work on ROS 1.0 and ROS 2.0 security is presented. In Chapter III, a security assessment is provided that describe the vulnerabilities of using ROS within unmanned systems. We highlight the potential solutions for these vulnerabilities and discuss the methods that are used in this thesis to mitigate certain attacks. In Chapter IV, we provide a discussion of the experimental setup and design. In Chapter V, we detail the results and analysis of our experimental results. In Chapter VI, we conclude the thesis and propose future research directions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED WORK

In this chapter, we discuss the related work that has been published by other researchers on ROS security. We also examine the ROS variants that have been developed over the years to overcome its various security shortcomings.

A. RESEARCH ON ROS 1.0 SECURITY

In recent years, research on ROS and its associated security measures have been investigated in the literature. The aim of most research in this realm is to identify possible solutions to ROS 1.0 security vulnerabilities. In [11], the authors study the behavior of ROS 1.0 when communications are encrypted using the Triple Data Encryption Standard (3DES). While 3DES is one of two options authorized for encryption by the National Institute of Standards and Technology (NIST), it is not as fast as other encryption algorithms, such as the Advanced Encryption Standard (AES). In [12], the authors study the use of Message Authentication Codes (MACs) in ROS 1.0 to achieve secure authentication for remote, non-native clients; however, neither of these approaches provide a holistic solution to ROS 1.0 security.

One group of academics from Cornell University sought to add security features to ROS at the application layer [13]. This action allows them to incorporate security features without having to manipulate ROS itself. They treat ROS as a black box and achieve security through the use of an authentication server [13]. This approach utilizes a four-step process beginning with authentication and key agreement, followed by the registration of publishing and subscribing nodes, and ending with the actual publishing of messages. The first step includes the association of specific topic(s) to a publisher node. The second and third steps involve the authentication of the subscriber and publisher nodes through the authentication server. Session keys are created in the final step to encrypt published messages. Their design provides for confidentiality, integrity, authentication, and authorization; however, given that their solution is not integrated with the ROS API, it cannot provide availability and non-repudiation [13].

Another group of researchers from the Institute for Robotics and Mechatronics in Austria proposed a different means to provide security to ROS [4]. Their solution is similar to SROS in that security elements are incorporated at the transport level. Their efforts were focused on securing node-to-node TCP and UDP communications. Through their technique, they are able to execute a methodical authorization scheme by looking at each available topic through an initial handshake where public key cryptography and certificates are the basis for mutual authentication and authorization [4]. Their technique also incorporates the use of symmetric encryption algorithms and MACs to guarantee confidentiality and integrity of communications [4]. The process begins with a remote procedure call known as XMLRPC from the subscriber to the publisher. The requested topic is subsequently verified, and then the publisher and subscriber exchange port information for connection, TCP or UDP, establishment [4]. A final handshake is used to secure the channel through an exchange of certificates between the client and server [4]. In addition, a challenge and response feature enables the creation of keys that are later used to encrypt transmitted messages [4].

B. RESEARCH ON SROS

As was mentioned earlier, SROS is an addition to the ROS API that is meant to provide cyber security measures. SROS introduces new security features that address encryption, access control and process profiles. Encryption in SROS is achieved through native Transport Layer Security (TLS) support for all socket-level communications through the use of X.509 certificates. These certificates are used in developing chains of trust, validating authenticity, and ensuring integrity. A key server is also maintained for key generation [5]. Access control is achieved through node restrictions and roles, auditing of the graph network through security logs, and through user-constructed access control policies [5]. Application security is achieved through process profiles. Node processes are hardened using Linux security modules in the kernel [5].

C. RESEARCH ON ROS 2.0 AND ROS-M

As mentioned earlier, ROS 2.0 implements its security measures using the DDS. The addition of DDS changes the overall communications architecture of ROS 1.0 from using messages, in a sometimes complex network programming paradigm, to a simpler and secure publish-subscribe method for sending/receiving data among nodes/topics. The addition of the DDS security component affords ROS 2.0 the ability to protect data that is actively being transmitted. DDS uses symmetric and asymmetric cryptography for data confidentiality and authentication. Hash functions in conjunction with pubic key infrastructure (PKI) are used to verify message integrity and non-repudiation [19]. Researchers with the United States Army Tank Automotive Research Development and Engineering Center (TARDEC) have begun to develop a militarized version of ROS to suit the needs of unmanned systems. The development of ROS-M is focused on the addition of military-relevant aspects to the ROS 2.0 architecture, namely additional simulation applications, the ability to perform cyber assurance checking, a means to store pertinent coding, and the creation of a training environment for service members [14]. TARDEC developed a phased approach to the development and implementation of ROS-M to Army unmanned systems. Phase III, which is a demonstration of the use of ROS-M on unmanned systems executing a maneuver over a prescribed course, was executed during their Industry Days event in April 2018 [14]. The security assessment portion of the demonstration included a successful man-in-the-middle attack on a simple ROS 1.0 application and the same man-in-the-middle attack failing to work on the ROS 2.0 version of the application for two separate unmanned ground vehicles, the Polaris MRZR and the John Deer M-Gator. Both vehicles were running ROS code developed for two separate military projects with a portion of the code updated to ROS 2.0 and all of the code registered as ROS-M components [14].

Although there has been a push by the Navy to extend and harden ROS 2.0, it has not been easily accepted into the fleet, in part due to a lack of proof of the quality of software (i.e., formal verification and validation), particularly in unmanned aerial systems.

THIS PAGE INTENTIONALLY LEFT BLANK

III. THREAT ASSESSMENT

Much work has been done in academia and industry to allay the network security concerns present within ROS. In this chapter, we outline the inherent vulnerabilities of ROS 1.0 and the potential use of ROS 2.0 to mitigate some of these problems. Given its future adaptability to military UxS, ROS 2.0 is viewed as a holistic tool for ROS security.

A. INHERENT ROS VULNERABILITIES

The structure of ROS 1.0 presents a set of vulnerabilities that can be exploited. ROS messages are sent between nodes in clear text, which means that they are not encrypted. Another feature of ROS that presents a problem is its anonymous graph-type structure. There are no procedures in place to validate nodes within the communications structure. Each connected node within the graph can subscribe to any topic that exists within the ROS instance. Nodes can act as publishers of information or as receivers of information. There is also an inherent vulnerability in how the ROS runtime process functions.

In the subsequent sections, we highlight the vulnerabilities that exist within the ROS 1.0 communications structure. We examine the exploits that can be executed against ROS and a UAV swarm. A description of each exploit is provided along with possible actions that can be taken to mitigate damage.

1. ROS Network

ROS network communications link vulnerabilties and mitigation techniques are provided in Table 1.

Table 1. ROS Network Communication Link Vulnerabilities and Mitigations

Exploit Name	Description	Possible Mitigations
Packet Sniffing	Information transmitted between ROS nodes are captured and analyzed	Strong encryption
Man-in-the-Middle	Node message traffic is intercepted, modified and passed on	Strong encryption and authentication
Rogue Nodes	Nodes that are capable of disturbing system operations by injecting commands about arbitrary topics [4]	Node specific authentication and authorization
Denial of Service	Nodes publishing large amounts of erroneous data which increases processing times at other nodes [4]	Node specific authentication and authorization [4]
Message Spoofing	Nodes publishing information to execute actions outside of the purview of valid user(s)	Node identification through authentication
Code Injection	Attackers that have gained access to the ROS network modifying the behavior of the system	Define access permissions that prevent system modification after running
Zero Day Exploits	Altering the ROS network permissions	Define access permissions that prevent system modification after running

2. UAV Swarm

The communications link used when operating in a swarm environment presents its own vulnerabilities. Specific UAV swarm communications link vulnerabilties and mitigation techniques are provided in Table 2.

Table 2. UAV Swarm Communication Link Vulnerabilities and Mitigations

Exploit Name	Description	Possible Mitigations
Eavesdropping	Information transmitted between ground control system (GCS) and UAV(s) intercepted	Strong encryption, directional antennas
Replay Attack	Intercepted packets recorded and broadcasted at a later time	Strong authentication and serialization
RC Hijack	Bad actor pairs with a UAV	Strong authentication between GCS and UAV

B. SUMMARY OF SECURITY VULNERABILITIES AND CHOSEN MITIGATION TECHNIQUES

Given the nature of our experimental environment, it is not feasible to implement mitigation techniques for every single type of risk previously identified. One can see from the given types of risks that a number of those listed can be mitigated with authentication schemes. The difficulty lies in the way the swarm communicates. Messages are broadcast omni-directionally. Encryption is required in order to keep sensitive information from being captured and exploited. Given the communications schemes utilized by the swarm in addition to the use of ROS, our system is particularly vulnerable to man-in-the-middle (MITM) attacks and replay attacks.

An individual with knowledge of ROS can introduce ROS messages into the drone to manipulate the actions of the vehicle. Moreover, a person with specific understanding of the ROS message fields within the given system graph can conceivably manipulate the drone with greater efficiency [15].

It must be noted that the addition of encryption and authentication will not completely protect the swarm from every type of attack. The communications channel that the swarm uses for its operation is still susceptible to denial-of-service (DoS). Ensuring channel availability is not part of this thesis work.

C. POTENTIAL ROS SPECIFIC SOLUTIONS

ROS 2.0, particularly its enhanced security features provided through DDS, appears to offer a ROS-specific solution to identified concerns with the ROS 1.0 platform. The ROS 2.0 DDS literature indicates that its features offer users a number of mitigation techniques to the exploits that have been identified in Tables 1 and 2; however, these security enhancements must be tested in order to verify their effectiveness. We chose to validate the ROS 2.0 DDS system against two attack vectors: a rogue-node style attack and a message-spoofing attack. These exploits are important first steps to validate ROS 2.0 DDS as a viable security solution for military centric operations. The processes within ROS 2.0 DDS that offer a means of node authentication and node permissions will be challenged through our testing.

IV. EXPERIMENTAL SETUP

ROS 2.0, through its available security enhancements, promises to provide solutions to many of the identified communications vulnerability exploits. In this chapter, we explain the simulation architecture that we used to test ROS 2.0 against specific cyber threats, specifically rogue-nodes and spoofing attacks.

A. SWARM SIMULATION

The goal of our simulations is to test the ability of ROS 2.0 to safeguard communications between each UAV and GCS. We tested ROS 2.0's ability to prevent threats posed by message spoofing and rogue-node attacks. Recall that ROS 2.0 is built atop the OMG DDS standard incorporating the eProsima Fast RTPS protocol. The security processes within this protocol are enabled when we test the viability of ROS 2.0 in effectively providing security through authentication, access control, and encryption. Our simulation setup required that we incorporate a bridge between ROS 2.0 and ROS 1.0, as the most recent version of ROS 2.0 (ROS 2.0 Ardent) does not support Gazebo. ROS 2.0 Ardent lacks the dependencies needed to function with Gazebo. Future releases of ROS 2.0 plan to address this issue, but a solution is not presently available without using a bridge. The bridge itself enables the exchange of messages between ROS 1.0 and ROS 2.0. The bridge acts as a ROS 1.0 node as well as a ROS 2.0 node at the same time and can, therefore, subscribe to messages in one ROS version and publish them into the other ROS version [16].

Our experiments were performed on a Mac Book Pro laptop with an Intel Core i7-3615QM Processor. The Mac operating system was replaced with a Linux operating system running Ubuntu 16.04 LTS. This particular version of Ubuntu was required for our chosen ROS API and simulation architecture to function properly. The PX4 Multi Vehicle Simulation was utilized in setting up the experiments. Within this simulation setup, ROS 1.0 Kinetic is used with PX4 and the Gazebo 9 simulator. PX4 autopilot allows a remotely piloted aircraft to be flown out of sight. The simulated drones, which in our simulation included three instances (i.e., three UAVs), are visualized in Gazebo. Our simulation utilizes a MAVROS MAVLink node in order to establish communication with PX4 [17].

MAVLink is a standard communications protocol for a UAV. MAVROS is an extension of MAVLink to ROS enabled devices. MAVROS allows for communications between the UAV autopilot and the GCS. [17]. QGroundControl v3.3.1 served as our GCS software. Through QGroundControl, the drone instances are armed, flightpath parameters are entered, and the flightpath is executed. The Gazebo drone simulation generates sensor data, including motor and actuator values, from its simulated world, which is then transmitted to PX4. PX4 communicates with ROS and the GCS to send drone telemetry information and to receive commands [17]. Figure 1 is a depiction of the MAVLink communications structure for the first UAV instance.

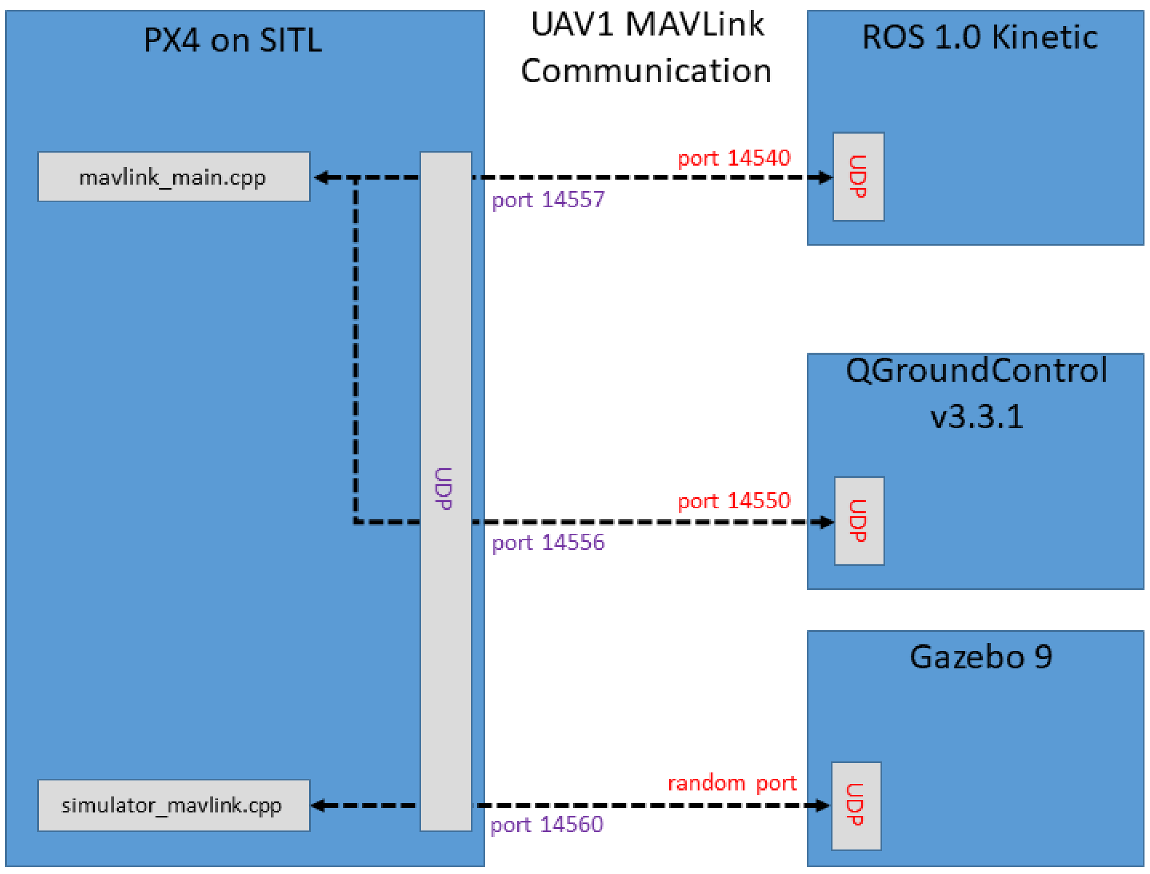


Figure 1. UAV1 Communications Architecture

Through the experiments, we tested the simulations against attempts to manipulate established MAVROS service nodes including the command arming, command landing and command takeoff nodes. The arming command is the instruction that is sent to an individual drone to turn on the drone itself and prepare it for a mission. The landing command specifies where a drone will execute a landing maneuver and can be used to immediately direct a drone to land at a specific location. The takeoff command is the directive that specifies how a takeoff is executed; this command can be used to immediately direct the drone to climb to a new location. In this thesis, we only consider the communication between the simulated drones and the GCS. The ROS 1.0 Kinetic was chosen for our experiments as it has been deemed the most functional version for use with ROS 2.0 and our chosen simulation visualization and control software. ROS 2.0 Ardent was released for public use in December of 2017 and was chosen for this thesis.

All available ROS nodes are displayed in the computation graph. The ROS computation graph is a GUI that displays the nodes that currently exist within the system. It visualizes the publish-subscribe relationship between ROS nodes. The ROS computation graph for our simulation is shown in Figure 2. When examining the figure, we can see distinct shapes and forms within the ROS graph. The oval in Figure 2 represents a ROS node. Publisher-subscriber associations are represented by directed edges [8]. It is here where we can see one of the targeted UAV MAVROS nodes for our experiment.

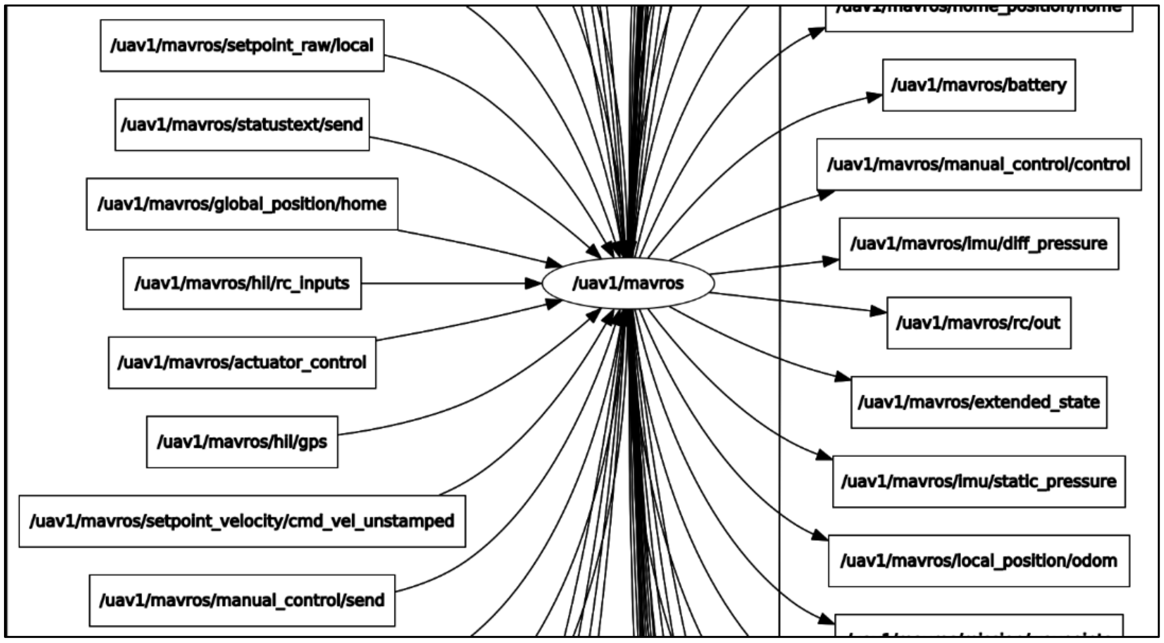
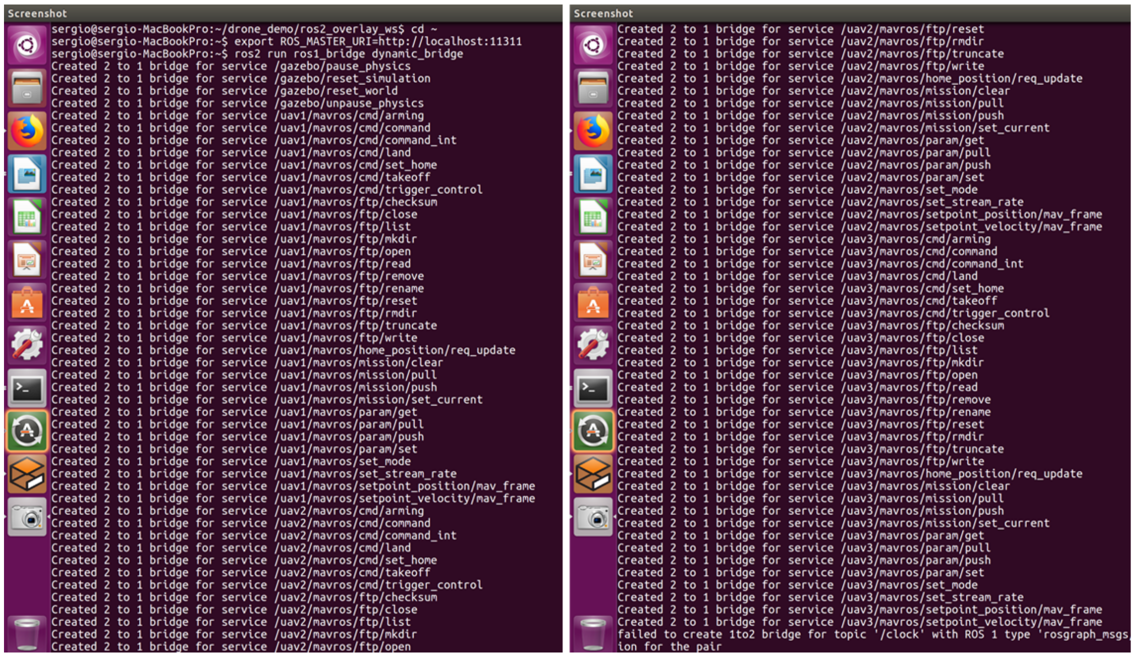

/uav1/mavros/setpoint_raw/local
/uav1/mavros/statustext/send
/uav1/mavros/global_position/home
/uav1/mavros/hil/rc_inputs
/uav1/mavros/actuator_control
/uav1/mavros/hil/gps
/uav1/mavros/setpoint_velocity/cmd_vel_unstamped
/uav1/mavros/manual_control/send
/uav1/mavros
/uav1/mavros/home_position/home
/uav1/mavros/battery
/uav1/mavros/manual_control/control
/uav1/mavros/imu/diff_pressure
/uav1/mavros/rc/out
/uav1/mavros/extended_state
/uav1/mavros/imu/static_pressure
/uav1/mavros/local_position/odom

Figure 2. Simulation ROS 1.0 Computation Graph Structure

A ROS 1.0 ROS 2.0 bridge is created so that the ROS 2.0 security features can be enabled in our drone simulation system. This bridge is built after the simulation launch file is run and the ROS 1.0 computation graph and node structures are present. The referenced launch file is provided in the Appendix. When creating the bridge, ROS 2.0 nodes and topics are generated to match those existing ROS 1.0 components. It is important that complete pairs be made so that the security features afforded to us from ROS 2.0 cover all communications within our ROS network. The bridge creation and two-to-one matching between ROS 2.0 and ROS 1.0 are shown in Figure 3.



```
sergio@sergio-MacBookPro:~/drone_demo/ros2_overlay_ws$ cd ~
sergio@sergio-MacBookPro:~$ export ROS_MASTER_URI=http://localhost:11311
sergio@sergio-MacBookPro:~$ ros2 run ros1_bridge dynamic_bridge
Created 2 to 1 bridge for service /gazebo/pause_physics
Created 2 to 1 bridge for service /gazebo/reset_simulation
Created 2 to 1 bridge for service /gazebo/reset_world
Created 2 to 1 bridge for service /gazebo/unpause_physics
Created 2 to 1 bridge for service /uav1/mavros/cmd/arming
Created 2 to 1 bridge for service /uav1/mavros/cmd/command
Created 2 to 1 bridge for service /uav1/mavros/cmd/command_int
Created 2 to 1 bridge for service /uav1/mavros/cmd/land
Created 2 to 1 bridge for service /uav1/mavros/cmd/set_home
Created 2 to 1 bridge for service /uav1/mavros/cmd/takeoff
Created 2 to 1 bridge for service /uav1/mavros/cmd/trigger_control
Created 2 to 1 bridge for service /uav1/mavros/ftp/checksum
Created 2 to 1 bridge for service /uav1/mavros/ftp/close
Created 2 to 1 bridge for service /uav1/mavros/ftp/list
Created 2 to 1 bridge for service /uav1/mavros/ftp/mkdir
Created 2 to 1 bridge for service /uav1/mavros/ftp/open
Created 2 to 1 bridge for service /uav1/mavros/ftp/read
Created 2 to 1 bridge for service /uav1/mavros/ftp/remove
Created 2 to 1 bridge for service /uav1/mavros/ftp/rename
Created 2 to 1 bridge for service /uav1/mavros/ftp/reset
Created 2 to 1 bridge for service /uav1/mavros/ftp/rmdir
Created 2 to 1 bridge for service /uav1/mavros/ftp/truncate
Created 2 to 1 bridge for service /uav1/mavros/ftp/write
Created 2 to 1 bridge for service /uav1/mavros/home_position/req_update
Created 2 to 1 bridge for service /uav1/mavros/mission/clear
Created 2 to 1 bridge for service /uav1/mavros/mission/pull
Created 2 to 1 bridge for service /uav1/mavros/mission/push
Created 2 to 1 bridge for service /uav1/mavros/mission/set_current
Created 2 to 1 bridge for service /uav1/mavros/param/get
Created 2 to 1 bridge for service /uav1/mavros/param/pull
Created 2 to 1 bridge for service /uav1/mavros/param/push
Created 2 to 1 bridge for service /uav1/mavros/param/set
Created 2 to 1 bridge for service /uav1/mavros/set_mode
Created 2 to 1 bridge for service /uav1/mavros/set_stream_rate
Created 2 to 1 bridge for service /uav1/mavros/setpoint_position/mav_frame
Created 2 to 1 bridge for service /uav1/mavros/setpoint_velocity/mav_frame
Created 2 to 1 bridge for service /uav2/mavros/cmd/arming
Created 2 to 1 bridge for service /uav2/mavros/cmd/command
Created 2 to 1 bridge for service /uav2/mavros/cmd/command_int
Created 2 to 1 bridge for service /uav2/mavros/cmd/land
Created 2 to 1 bridge for service /uav2/mavros/cmd/set_home
Created 2 to 1 bridge for service /uav2/mavros/cmd/takeoff
Created 2 to 1 bridge for service /uav2/mavros/cmd/trigger_control
Created 2 to 1 bridge for service /uav2/mavros/ftp/checksum
Created 2 to 1 bridge for service /uav2/mavros/ftp/close
Created 2 to 1 bridge for service /uav2/mavros/ftp/list
Created 2 to 1 bridge for service /uav2/mavros/ftp/mkdir
Created 2 to 1 bridge for service /uav2/mavros/ftp/open
```

```
Created 2 to 1 bridge for service /uav2/mavros/ftp/reset
Created 2 to 1 bridge for service /uav2/mavros/ftp/rmdir
Created 2 to 1 bridge for service /uav2/mavros/ftp/truncate
Created 2 to 1 bridge for service /uav2/mavros/ftp/write
Created 2 to 1 bridge for service /uav2/mavros/home_position/req_update
Created 2 to 1 bridge for service /uav2/mavros/mission/clear
Created 2 to 1 bridge for service /uav2/mavros/mission/pull
Created 2 to 1 bridge for service /uav2/mavros/mission/push
Created 2 to 1 bridge for service /uav2/mavros/mission/set_current
Created 2 to 1 bridge for service /uav2/mavros/param/get
Created 2 to 1 bridge for service /uav2/mavros/param/pull
Created 2 to 1 bridge for service /uav2/mavros/param/push
Created 2 to 1 bridge for service /uav2/mavros/param/set
Created 2 to 1 bridge for service /uav2/mavros/set_mode
Created 2 to 1 bridge for service /uav2/mavros/set_stream_rate
Created 2 to 1 bridge for service /uav2/mavros/setpoint_position/mav_frame
Created 2 to 1 bridge for service /uav2/mavros/setpoint_velocity/mav_frame
Created 2 to 1 bridge for service /uav3/mavros/cmd/arming
Created 2 to 1 bridge for service /uav3/mavros/cmd/command
Created 2 to 1 bridge for service /uav3/mavros/cmd/command_int
Created 2 to 1 bridge for service /uav3/mavros/cmd/land
Created 2 to 1 bridge for service /uav3/mavros/cmd/set_home
Created 2 to 1 bridge for service /uav3/mavros/cmd/takeoff
Created 2 to 1 bridge for service /uav3/mavros/cmd/trigger_control
Created 2 to 1 bridge for service /uav3/mavros/ftp/checksum
Created 2 to 1 bridge for service /uav3/mavros/ftp/close
Created 2 to 1 bridge for service /uav3/mavros/ftp/list
Created 2 to 1 bridge for service /uav3/mavros/ftp/mkdir
Created 2 to 1 bridge for service /uav3/mavros/ftp/open
Created 2 to 1 bridge for service /uav3/mavros/ftp/read
Created 2 to 1 bridge for service /uav3/mavros/ftp/remove
Created 2 to 1 bridge for service /uav3/mavros/ftp/rename
Created 2 to 1 bridge for service /uav3/mavros/ftp/reset
Created 2 to 1 bridge for service /uav3/mavros/ftp/rmdir
Created 2 to 1 bridge for service /uav3/mavros/ftp/truncate
Created 2 to 1 bridge for service /uav3/mavros/ftp/write
Created 2 to 1 bridge for service /uav3/mavros/home_position/req_update
Created 2 to 1 bridge for service /uav3/mavros/mission/clear
Created 2 to 1 bridge for service /uav3/mavros/mission/pull
Created 2 to 1 bridge for service /uav3/mavros/mission/push
Created 2 to 1 bridge for service /uav3/mavros/mission/set_current
Created 2 to 1 bridge for service /uav3/mavros/param/get
Created 2 to 1 bridge for service /uav3/mavros/param/pull
Created 2 to 1 bridge for service /uav3/mavros/param/push
Created 2 to 1 bridge for service /uav3/mavros/param/set
Created 2 to 1 bridge for service /uav3/mavros/set_mode
Created 2 to 1 bridge for service /uav3/mavros/set_stream_rate
Created 2 to 1 bridge for service /uav3/mavros/setpoint_position/mav_frame
Created 2 to 1 bridge for service /uav3/mavros/setpoint_velocity/mav_frame
failed to create 1to2 bridge for topic '/clock' with ROS 1 type 'rosgraph_msgs
ion for the pair
```

Figure 3. ROS Bridge Creation

B. ROS 2.0 SECURITY ARCHITECTURE

Our simulation was built with a ROS 1.0 framework; however, we take advantage of ROS 2.0 security features by creating a communications bridge between both APIs. As discussed previously, ROS 2.0 provides encryption, authentication, and process profile features by activating the eProsima Fast RTPS protocol, which is incorporated within the DDS standard upon which that ROS 2.0 is built. Security through the implementation of Fast RTPS is provided through three distinct methods. Fast RTPS authenticates remote participants, provides access to verified entities, and encrypts transmitted data [9]. Each level of security is achieved through the activation of the following features that are provided within the DDS middleware.

1. Authentication

This feature provides for the authentication of ROS nodes. The Fast RTPS authentication plugin, when activated, authenticates detected ROS nodes [9]. This process continues until all ROS nodes are authenticated. Node authentication is achieved through use of the Elliptic Curve Digital Signature Algorithm (ECDSA), while a shared secret key is created using Elliptic Curve Diffie-Hellman (ECDH) for use in communications encryption [9]. PKI, utilizing a pre-configured and communal Certificate Authority (CA), is incorporated by the authentication feature [18].

2. Access Control

This attribute provides validation of ROS node permissions after a remote participant is authenticated. Participants are assigned permissions after they are authenticated. These permissions are validated and enforced [18]. Node permissions are confirmed and applied and the access rights that are assigned to each node over available system resources are defined. Access control is achieved through the creation of a permissions document signed by the shared CA [18]. The Domain Governance and Permissions document is signed by an X.509 permissions certificate. It is an XML document that defines how the environment is secured [9].

3. Cryptographic

This feature provides encryption and is applied to communications messages exchanged between ROS nodes. This feature is configured by the access control plugin. The cryptography feature utilizes Advanced Encryption Standard in Galois Counter Mode (AES-GCM) for encryption and AES Galois Message Authentication Code (AES-GMAC) for the authentication of messages [18].

These security features are enabled only after the simulation architecture has been created on both the ROS 1.0 and ROS 2.0 side. By performing this step last, we ensure that only existing ROS artifacts, namely the established system and nodal relationships, are incorporated into the access control documents. This ensures legitimacy of all aspects of our system.

V. RESULTS AND ANALYSIS

A series of simulations under three different conditions was conducted to demonstrate the strength of ROS 2.0 in the face of specific threat vectors. In this chapter, we discuss our simulation results and their effectiveness in thwarting rogue-node and message spoofing attacks. We also highlight the significance of these results in validating ROS 2.0 from a security perspective.

A. SIMULATION RESULTS

1. Baseline

In order to establish a baseline for the simulations, a definitive flightpath was loaded into each of the three simulated UAVs. The flightpath was designed to mirror a reconnaissance mission over a defined route, which is a likely mission for a small UAV. The route was determined based on the geographic features provided by the QGroundControl software. The UAVs executed this given flightpath under three specific conditions:

- Condition 1: The simulation is run with ROS 1.0.
- Condition 2: The simulation is run with the ROS 1.0 ROS 2.0 Bridge devoid of any security features.
- Condition 3: The simulation is run with the ROS 1.0 ROS 2.0 Bridge with security features enabled.

We ran the simulation through ten individual trials for each of the given conditions. Our simulation baseline trials were conducted without any malicious activity. Time was recorded from the moment the first UAV began its ascent to the moment the third UAV landed safely on the ground. The results for our attack-free trials of the simulated experiment are provided in Table 3. We refer to these results as our baseline throughout the remainder of the thesis.

Table 3. Simulation Baseline Results

Simulation Run Times (min:sec)											
Conditions	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	~Avg
ROS 1.0	2:30	2:20	2:31	2:17	2:28	2:44	2:39	2:27	2:24	2:40	2:30
ROS 1.0/ROS2.0 Bridge	2:33	2:32	2:23	2:19	2:30	2:31	2:47	2:41	2:27	2:42	2:32
Security Enabled ROS 1.0/ROS 2.0 Bridge	2:33	2:31	2:28	2:35	2:30	2:40	2:28	2:26	2:45	2:44	2:34

As can be seen in Table 3, our results show that the addition of the ROS 1.0 ROS 2.0 Bridge added a negligible amount of time to our simulation when compared to the simulation running on only ROS 1.0. On average, two additional seconds were needed to complete the entire flight path under the second condition. When the security features are enabled on the bridge, more time is required to execute the flight path. On average, an additional four seconds were needed to complete the mission under the third condition compared to the first condition. The additional time required by the simulation to complete the prescribed flight path under the second and third conditions is measurable and appears to be adding a delay overhead.

2. Drone Disabling

The next set of trials involved a rogue node completely disabling an individual UAV. The rogue node accesses the MAVROS command arming service and directs it to shut down the targeted drone. This action caused the drone to instantly shutoff its engines and resulted in the drone crashing to the ground. UAV1, UAV2, and UAV3 represent the targeted nodes/drones. The same conditions used as part of the simulation baseline were applied to these trials. Each individual UAV was targeted under each condition for a total of ninety trials. The results are provided in Table 4.

Table 4. Drone Disabling Simulation Results

Time to Disable Drone (min:sec)											
Conditions	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	~Avg
ROS 1.0											
UAV1	0:40	0:41	0:42	0:40	0:41	0:42	0:42	0:40	0:42	0:41	0:41.1
UAV2	0:41	0:41	0:42	0:41	0:41	0:41	0:42	0:41	0:40	0:42	0:41.2
UAV3	0:42	0:42	0:40	0:41	0:42	0:40	0:41	0:40	0:41	0:42	0:41.1
ROS 1.0/ROS2.0 Bridge											
UAV1	0:44	0:42	0:43	0:42	0:43	0:42	0:42	0:45	0:44	0:45	0:43.2
UAV2	0:42	0:42	0:43	0:44	0:45	0:43	0:41	0:44	0:43	0:45	0:43.2
UAV3	0:44	0:42	0:43	0:44	0:45	0:43	0:41	0:44	0:43	0:44	0:43.3
Security Enabled ROS 1.0/ROS 2.0 Bridge											
UAV1	2:54	3:02	2:48	3:01	3:09	2:47	3:06	3:04	3:02	2:53	2:59
UAV2	2:55	3:01	2:47	3:03	3:10	2:46	3:07	3:05	3:03	2:54	2:59
UAV3	2:52	3:03	2:49	3:02	3:11	2:48	3:08	3:05	3:05	2:55	3:00

Under the first and second conditions, the numbers listed for each trial show the time at which the entire drone swarm was successfully in the air flying. The attack itself was near instantaneous. As soon as the attack was executed, the targeted drone fell out of the sky. We can see that between the first and second conditions, it took on average two additional seconds to get the drones in the air when the bridge was enabled; thus, a small delay overhead is incurred when the bridge is enabled.

For the third condition (Bridge Secured), the numbers represent the total time that the UAVs took to complete the flight path. In each of the trials, the attempted attacks failed; however, repeated attempts to disable an individual drone added a substantial amount of time to the flight path. When compared to the baseline simulation, the additional time amounted to an extra 25 seconds. Considering that the entire flight time for the baseline trials was on average two minutes and 59 seconds, the additional flight time amounted to a 16% increase. In addition, these results are evidence of ROS 1.0's inability to guard against an attack when compared to ROS 2.0.

3. Drone Forced Landing

The next set of trials involved a message-spoofing attack. In this attack, the creation of a malicious node takes control of a UAV and forces it to land. The MAVROS command-landing service was accessed. The rogue node directed the targeted drone to land at a prescribed location at the time the command was sent. The same conditions used as part of the simulation baseline were applied to these trials. Each individual UAV, UAV1 through UAV3, was targeted under each condition for a total of ninety trials. The timing results under this attack are shown in Table 5.

Table 5. Drone Forced Landing Results

Time to Force Land Drone (min:sec)										
Conditions	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	~Avg
ROS 1.0											
UAV1	0:42	0:41	0:41	0:42	0:43	0:42	0:44	0:42	0:41	0:42	0:42.0
UAV2	0:42	0:43	0:41	0:42	0:43	0:41	0:43	0:42	0:42	0:43	0:42.2
UAV3	0:43	0:42	0:42	0:43	0:42	0:43	0:42	0:43	0:41	0:42	0:42.3
ROS 1.0/ROS2.0 Bridge											
UAV1	0:45	0:43	0:43	0:44	0:45	0:43	0:44	0:45	0:44	0:45	0:44.1
UAV2	0:44	0:45	0:44	0:45	0:44	0:45	0:42	0:44	0:43	0:46	0:44.2
UAV3	0:45	0:44	0:43	0:44	0:45	0:44	0:43	0:44	0:45	0:44	0:44.1
Security Enabled ROS 1.0/ROS 2.0 Bridge											
UAV1	2:55	3:03	2:57	3:03	3:07	2:51	3:04	3:05	3:03	3:04	3:01
UAV2	3:07	3:03	2:56	3:05	3:01	3:06	2:55	2:53	3:00	3:05	3:01
UAV3	2:53	3:01	2:51	3:03	3:09	2:50	3:05	3:07	3:02	2:58	3:00

Under the first and second conditions, the numbers listed for each trial show the time at which the entire drone swarm was successfully in the air flying. The effects of the attacks themselves, as in the previous trial, were near immediate. As soon as the attack was executed, the targeted drone began to execute a landing maneuver. The execution of the landing itself took just over one second to execute as the drone was flying at an altitude of only ten meters. Again, we observe a two-second delta between trials under the first and second condition. The cost in time to run the bridge is still present.

With the third condition, the numbers represent the total time that the UAVs took to complete the flight path. In each of the trials, the attempted attacks continued to fail as before. As was observed in the previous trials, the repeated attempts to disable an individual drone added a substantial amount of time to the flight path. When compared to the baseline simulation, the additional time amounted to an extra 27 seconds. Considering that the entire flight time for the baseline trials was on average two minutes and 59 seconds, the additional flight time amounts to a 17.5% increase. As was the case before, ROS 1.0 fails at protecting against this threat, whereas ROS 2.0 is successful but incurs a tradeoff in delay.

4. Drone Forced Climb

The last set of trials also involved a message spoofing attack. In this attack, the creation of a malicious node takes control of a UAV and forces it to gain altitude. The MAVROS command takeoff service was accessed. The rogue node directed the targeted drone to climb to a prescribed location at the time the command was sent. The same conditions used as part of the simulation baseline were applied to these trials. Each individual UAV was targeted under each condition for a total of ninety trials. The timing results under this attack are shown in Table 6.

Table 6. Drone Forced Climb Results

Time to Force Climb (min:sec)											
Conditions	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	~Avg
ROS 1.0											
UAV1	0:41	0:42	0:42	0:40	0:41	0:40	0:42	0:43	0:41	0:42	0:41.4
UAV2	0:43	0:41	0:43	0:41	0:42	0:41	0:42	0:40	0:42	0:41	0:41.6
UAV3	0:42	0:40	0:44	0:40	0:42	0:41	0:42	0:43	0:41	0:40	0:41.5
ROS 1.0/ROS2.0 Bridge											
UAV1	0:43	0:45	0:43	0:42	0:45	0:43	0:44	0:43	0:42	0:45	0:43.5
UAV2	0:44	0:43	0:44	0:43	0:43	0:44	0:42	0:45	0:42	0:44	0:43.4
UAV3	0:44	0:42	0:45	0:41	0:45	0:43	0:44	0:46	0:44	0:41	0:43.5
Security Enabled ROS 1.0/ROS 2.0 Bridge											
UAV1	2:56	3:02	2:55	3:01	3:05	2:53	3:05	3:02	3:00	3:03	3:00
UAV2	3:04	3:04	3:01	3:02	2:58	2:58	3:03	3:01	2:55	3:05	3:01
UAV3	2:54	3:07	2:55	2:52	3:02	3:02	3:01	3:07	3:06	2:59	3:00

The numbers recorded under the first and second condition continue to demonstrate the time required to have the entire drone swarm airborne. The effects of the attacks themselves, as in the previous first trial, were again instantaneous. As soon as the attack was executed, the targeted drone began to execute a climbing maneuver. As soon as the attack was executed, the targeted drone began to climb and gain in altitude. The additional two seconds are still present between trials executed under the first and second condition. Again, the cost in time to run the bridge is still present.

As before, under the third condition, the recorded data represents the total time that the UAVs took to run through the prescribed mission. In each of the trials, the attempted attacks continued to fail as before. As was observed in the previous two attack experiments, the repeated attempts to disable an individual drone added a substantial amount of time to the flight path. When compared to the baseline simulation, the additional time amounted to an extra 26 seconds. Considering that the entire flight time for the baseline trials was on average two minutes and 59 seconds, the additional flight time amounts to just under a 17% increase.

5. Summary of Findings

We see from examination of the simulation results under the third condition, where security was enabled on the ROS 2.0 ROS 1.0 Bridge, proved to be effective at mitigating each attack vector. Given these observations, ROS 2.0 DDS works well in mitigating basic attacks; however, the effectiveness of this setup is inhibited by a significant latency overhead. It is our belief that implementation of the bridge was the prime factor in increasing the delay in flight time. As ROS 2.0 develops, the need for the bridged approach may no longer be required. Future ROS 2.0 variants that can directly function with Gazebo will likely reduce a portion of the delay overhead we observed.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. SUMMARY

The prominence of UxS and their utility as a force multiplier for the United States military will endure. These systems will continue to change the face of warfare and must be guarded against network and cyber threats. In this thesis, we studied the viability of using ROS 2.0 for UxS missions. We tested the ability of ROS 2.0 to mitigate against basic attacks in order to begin the formal verification process for transition to the fleet.

Through our UAV swarm configuration and communications architecture, we demonstrated the very real threat that exists to UxS that are constructed and operated devoid of any security architecture. By enabling given security enhancements available to the ROS through the ROS 2.0 API, we were able to demonstrate that these added features are capable of thwarting attacks against a small UAV swarm. The security features were effective in stifling the rogue-node and message spoofing attacks, but there was a measureable cost to the drone operation in terms of delay. A tradeoff exists between the use of ROS 2.0 and the latency overhead that is induced.

Operating a UxS, especially in a swarming configuration, without any network communications security is much costlier than the added time to conduct operations. The loss of a UxS to an adversary through the applied attack vectors would incur incredible replacement costs due to downed systems and breach of potentially sensitive information due to capture of the platform. It is also the expectation that in the future, ROS 2.0 will be functional without the use of a bridge. This may reduce the latency overhead.

B. FUTURE WORK

In this thesis, authentication, encryption, and access control features were enabled to provide security to our swarm simulation. Application of these features were limited to our simulation environment and limited to the classes of attack trajectories against which we chose to test. Further research and testing is needed to fully gauge the effectiveness of ROS 2.0 and DDS.

1. Additional Vulnerability Testing

Our experiments can be taken further by testing its viability against all of identified security vulnerabilities. There are over twenty other MAVROS specific nodes beyond the three that we tested that exist for each UAV instance that can exploited. Additional network communication vulnerabilities and mitigations should be explored.

2. Measured Power Use

Throughout our simulations, we measured an increased amount of time required to complete the mission tasked to the drone swarm. The addition of the bridge came with a cost, and the use of the security features came with a much higher cost. One could measure this cost in time against the physical tax that it takes against the UAVs power stores.

3. Testing against Networked UAVs

In this thesis, the testing of ROS 2.0 focused solely on communications between an individual UAV and the GCS. Testing of ROS 2.0 on communications between UAVs during a specific flight maneuver is required.

4. Adaption to a Physical Swarm

Eventually, the testing of ROS 2.0’s cybersecurity features should be carried out on hardware, moving beyond the simulated environment and into real world testing. Testing and basic experimentation on actual UAV assets to support simulated results would be beneficial.

APPENDIX. MULTIPLE DRONE SIMULATION FILE

The following ROS launch file is what used in this thesis to create our simulation environment. This file launches the Gazebo 9 simulation software and spawns three iris quadcopter drones with requisite MAVROS and PX SITL architecture. Each drone is assigned specific and unique UDP ports required for the MAVLink communications architecture.

```
<?xml version="1.0"?>
<launch>
 <!-- MAVROS posix SITL environment launch script -->
 <!-- launches Gazebo environment and 3x: MAVROS, PX4 SITL, and spawns vehicle --
>
 <!-- vehicle model and world -->
 <arg name="est" default="ekf2"/>
 <arg name="vehicle" default="iris"/>
 <arg name="world" default="$(find mavlink_sitl_gazebo)/worlds/empty.world"/>
 <!-- gazebo configs -->
 <arg name="gui" default="true"/>
 <arg name="debug" default="false"/>
 <arg name="verbose" default="false"/>
 <arg name="paused" default="false"/>
 <!-- Gazebo sim -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
 <arg name="gui" value="$(arg gui)"/>
 <arg name="world_name" value="$(arg world)"/>
 <arg name="debug" value="$(arg debug)"/>
 <arg name="verbose" value="$(arg verbose)"/>
 <arg name="paused" value="$(arg paused)"/>
 </include>
 <!-- UAV1 -->
 <group ns="uav1">
 <!-- MAVROS and vehicle configs -->
 <arg name="ID" value="1"/>
 <arg name="fcu_url" default="udp://:14540@localhost:14557"/>
 <!-- PX4 SITL and vehicle spawn -->
 <include file="$(find px4)/launch/single_vehicle_spawn.launch">
 <arg name="x" value="0"/>
 <arg name="y" value="0"/>
 <arg name="z" value="0"/>
 <arg name="R" value="0"/>
 <arg name="P" value="0"/>
 <arg name="Y" value="0"/>
 <arg name="vehicle" value="$(arg vehicle)"/>
```

```
<arg name="rcS" value="$(find px4)/posix-configs/SITL/init/$(arg est)/$(arg
vehicle)_$(arg ID)"/>
<arg name="mavlink_udp_port" value="14560"/>
<arg name="ID" value="$(arg ID)"/>
</include>
<!-- MAVROS -->
<include file="$(find mavros)/launch/px4.launch">
<arg name="fcu_url" value="$(arg fcu_url)"/>
<arg name="gcs_url" value=""/>
<arg name="tgt_system" value="$(arg ID)"/>
<arg name="tgt_component" value="1"/>
</include>
</group>
<!-- UAV2 -->
<group ns="uav2">
<!-- MAVROS and vehicle configs -->
<arg name="ID" value="2"/>
<arg name="fcu_url" default="udp://:14541@localhost:14559"/>
<!-- PX4 SITL and vehicle spawn -->
<include file="$(find px4)/launch/single_vehicle_spawn.launch">
<arg name="x" value="1"/>
<arg name="y" value="0"/>
<arg name="z" value="0"/>
<arg name="R" value="0"/>
<arg name="P" value="0"/>
<arg name="Y" value="0"/>
<arg name="vehicle" value="$(arg vehicle)"/>
<arg name="rcS" value="$(find px4)/posix-configs/SITL/init/$(arg est)/$(arg
vehicle)_$(arg ID)"/>
<arg name="mavlink_udp_port" value="14562"/>
<arg name="ID" value="$(arg ID)"/>
</include>
<!-- MAVROS -->
<include file="$(find mavros)/launch/px4.launch">
<arg name="fcu_url" value="$(arg fcu_url)"/>
<arg name="gcs_url" value=""/>
<arg name="tgt_system" value="$(arg ID)"/>
<arg name="tgt_component" value="1"/>
</include>
</group>
<!-- UAV3 -->
<group ns="uav3">
<!-- MAVROS and vehicle configs -->
<arg name="ID" value="3"/>
<arg name="fcu_url" default="udp://:14551@localhost:14569"/>
```

```
 <!-- PX4 SITL and vehicle spawn -->
 <include file="$(find px4)/launch/single_vehicle_spawn.launch">
 <arg name="x" value="2"/>
 <arg name="y" value="0"/>
 <arg name="z" value="0"/>
 <arg name="R" value="0"/>
 <arg name="P" value="0"/>
 <arg name="Y" value="0"/>
 <arg name="vehicle" value="$(arg vehicle)"/>
 <arg name="rcS" value="$(find px4)/posix-configs/SITL/init/$(arg est)/$(arg
vehicle)_$(arg ID)"/>
 <arg name="mavlink_udp_port" value="14572"/>
 <arg name="ID" value="$(arg ID)"/>
 </include>
 <!-- MAVROS -->
 <include file="$(find mavros)/launch/px4.launch">
 <arg name="fcu_url" value="$(arg fcu_url)"/>
 <arg name="gcs_url" value=""/>
 <arg name="tgt_system" value="$(arg ID)"/>
 <arg name="tgt_component" value="1"/>
 </include>
 </group>
</launch>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Thompson and P. Thulasiraman, “Confidential and authenticated communications in a large fixed wing UAV swarm,” M.S. thesis, Department of Electrical and Computer Engineering, NPS, Monterey, CA, USA, Dec. 2016.

[2] L. Chen, Z. Wei, F. Zhao and T. Tao, “Development of a virtual teaching pendant system for serial robots based on ROS-I,” *in Proc. of IEEE International Conference on Cybernetics and Intelligent Systems and IEEE Conference on Robotics, Automation and Mechatronics*, pp. 720–724, Nov. 2017. [Online]. doi:10.1109/ICCIS.2017.8274867

[3] Army, “TARDEC 30-year strategy value stream analysis,” Sep. 27, 2016. [Online]. Available: https://www.army.mil/article/175820/tardec_30_year_strategy_value_stream_anal ysis

[4] B. Breiling, B. Dieber and P. Schartner, “Secure communication for the robot operating system,” *Annual IEEE International Systems Conference (SysCon)*, pp. 1-6, Apr. 2017. [Online]. doi:10.1109/SYSCON.2017.7934755

[5] R. White, H. I. Christensen and M. Quigley, “SROS: Securing ROS over the wire in the graph and through the kernel,” *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, Nov. 2016, [Online]. Available: https://arxiv.org/pdf/1611.07060.pdf

[6] A. Singhal, P. Pallav, N. Kejriwal, S. Choudhury, S. Kumar and R. Sinha, “Managing a fleet of autonomous mobile robots (AMR) using cloud robotics platform,” *European Conference on Mobile Robots (ECMR)*, pp. 1–6, Sep. 2017. [Online]. doi: 10.1109/ECMR.2017.8098721

[7] K. W. Wong and H. Kress-Gazit, “From high-level task specification to Robot Operating System (ROS) implementation,” *First IEEE International Conference on Robotic Computing (IRC)*, pp. 188–195, Apr. 2017. [Online]. doi: 10.1109/IRC.2017.18

[8] J. M. OKane, *A Gentle Introduction to ROS*. Columbia, SC: University of South Carolina, 2014. [Online] Available: https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf

[9] “Security,” Security - Fast RTPS 1.6.0 documentation, 2018. [Online]. Available: http://docs.eprosima.com/en/latest/security.html

[10] Github. “ros2/ros2,” Jun. 2018. [Online]. Available: https://github.com/ros2/ros2/wiki/DDS-and-ROS-middleware-implementations

[11] F. Rodriguez et al. “Cybersecurity in Autonomous Systems: Evaluating the Performance of Hardening ROS” *IEEE Workshop of Physical Agents*, pp. 1–7, Jun. 2016. [Online]. Available: https://pdfs.semanticscholar.org/ae27/f3cc833f3392a2929199261540ae53e4bcab.pdf

[12] R. Toris, C. Shue and S. Chernova, “Message Authentication codes for secure remote non-native client connections to ROS enabled robots,” *IEEE International Conference on Technologies for Practical Robot Applications*, pp. 1–6, Apr. 2014. [Online]. doi: 10.1109/TePRA.2014.6869141

[13] B. Dieber, S. Kacianka, S. Rass and P. Schartner, “Application-level security for ROS-based applications,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4477–4482, Oct. 2016. [Online]. doi: 10.1109/IROS.2016.7759659

[14] Army, “Industry Days draws more than 500,” Apr. 25, 2018. [Online]. Available: https://tardec.army.mil/#article/7

[15] J. Mcclean, C. Stull, C. Farrar and D. Mascareñas, “A preliminary cyber-physical security assessment of the Robot Operating System (ROS),” *Proc. SPIE 8741, Unmanned Systems Technology XV, 874110*, May 17, 2013. [Online]. doi: 10.1117/12.2016189

[16] Github. “ros2/ros1_bridge,” 2018. [Online]. Available: https://github.com/ros2/ros1_bridge/blob/master/doc/index.rst

[17] “ROS with Gazebo Simulation,” PX4 Developer Guide. [Online]. Available: https://dev.px4.io/en/simulation/ros_interface.html

[18] Object Management Group. “About the DDS Security Specification Version 1.0,” 2018. [Online]. Available: https://www.omg.org/spec/DDS-SECURITY/1.0/#specification-metadata

[19] V. Diluoffo, W. R. Michalson and B. Sunar, “Robot Operating System 2: The need for a holistic security approach to robotic architectures,” *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, May 3, 2018. [Online]. doi:10.1177/1729881418770011

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California