CS Capstone Final Report

June 13, 2017

Security for Robotics

Prepared for

Oregon State University

Vedanth Narayanan

Prepared by

Group 50

RoboSec

Emily Longman
Zach Rogers
Dominic Giacoppe

Table of Contents

Introduction

Our project was proposed by Vedanth Narayanan, a graduate student in CS at OSU, as a way to generate material for his masters thesis. For his thesis he plans to do something with security in ROS, the Robotics Operating System and henceforth referred to as ROS for the rest of this document, and our work was to discover and document security vulnerabilities for him to use as testing material later, along with identifying what parts of ROS were the most vulnerable overall. Vedanth was also our client for the project, and we, the team, were, in no particular order, Dominic Giacoppe, Emily Longman, and Zachary Rogers. The team had no defined roles at the outset, but over time Emily became the primary research design and compiler, Zach was the hardware specialist before the team decided to drop hardware from consideration from the project, and Dominic was the ROS expert and was the go-between for the team and Vedanth. Vedanth produced a small bit of code for the team, but on the whole took a supervisor position for the duration of the project, mostly providing ideas for what to explore next and detailing what he expected from the team at any given time.

SOFTWARE REQUIREMENTS SPECIFICATION

for

SecurityforRobotics

Version 1.0 Final approved

Prepared by
EmilyLongman, ZachRogers, DominicGiacoppe

June 11, 2017

Contents

1 Signatures

Sponsor Date

Group Member Date

Group Member Date

Group Member Date

2 Abstract

In drones and other networked robotics there is a broad array of security vulnerabilities that can be leveraged in an attack, leaving the potential for disaster. To attempt to prevent and mitigate these we evaluated ROS on a drone to find security holes and document them. The different vulnerabilities found were categorized into malware, sensor hacks, network and control channel attacks, and physical attacks. For some of these attacks were able to implement solutions, which were also documented. These findings and any solutions will be added to an ongoing academic effort to make robotics more secure.

3 Introduction

3.1 Purpose

To define the requirements and deliverables for Group 50's capstone project to our sponsor, Vedanth Narayanan.

3.2 Scope

We are to find security vulnerabilities in ROS/SROS, document these vulnerabilities, and if possible, produce patches for anything we find. Any patches produced will be submitted to the ROS project. Our testing will be focused around ROS/SROS running on a drone, and we will see if we can compromise that drone based on our findings.

3.3 Definitions

- Vulnerability: Any exploitable piece of code or system that would allow unauthorized users to interact with/damage/control the system, especially in a malicious manner.
- ROS: Robot Operating system, as found at link
- SROS: Secure ROS; a project based on ROS with the goal of implementing various security standards.
- Reliability: Consistently performs according to its specifications.
- Integrity: Maintaining and assuring the accuracy and consistency of data.
- Authentication: Any process where a system verifies the identity of a user who wishes to access it.

3.4 Overview

This document gives an overall agenda for our research. It is divided into 3 main sections: Section 1 is the introduction and purpose of this document, along with some acronym definitions. Section 2 contains the overall description of the goals we hope to achieve in our research. Section 3 lists our specific goals for our research.

4 Overall Description

4.1 Product Perspective

All software developed by Group 50 should have 2 objectives: 1. Fixing a specific, known vulnerability in ROS/SROS 2. Be lightweight enough that the implementation doesn't drastically affect the overall operation of the robot. Ideally, any code produced would be later incorporated into ROS itself, and not an external layer or program.

4.2 Product Functions

The main product we will be producing is research documentation on the potential security exploits in ROS. This will hopefully be used in future research and development of a more secure ROS. If we are able to exploit a vulnerability and create a functional patch for it, the changes will be submitted to the SROS project.

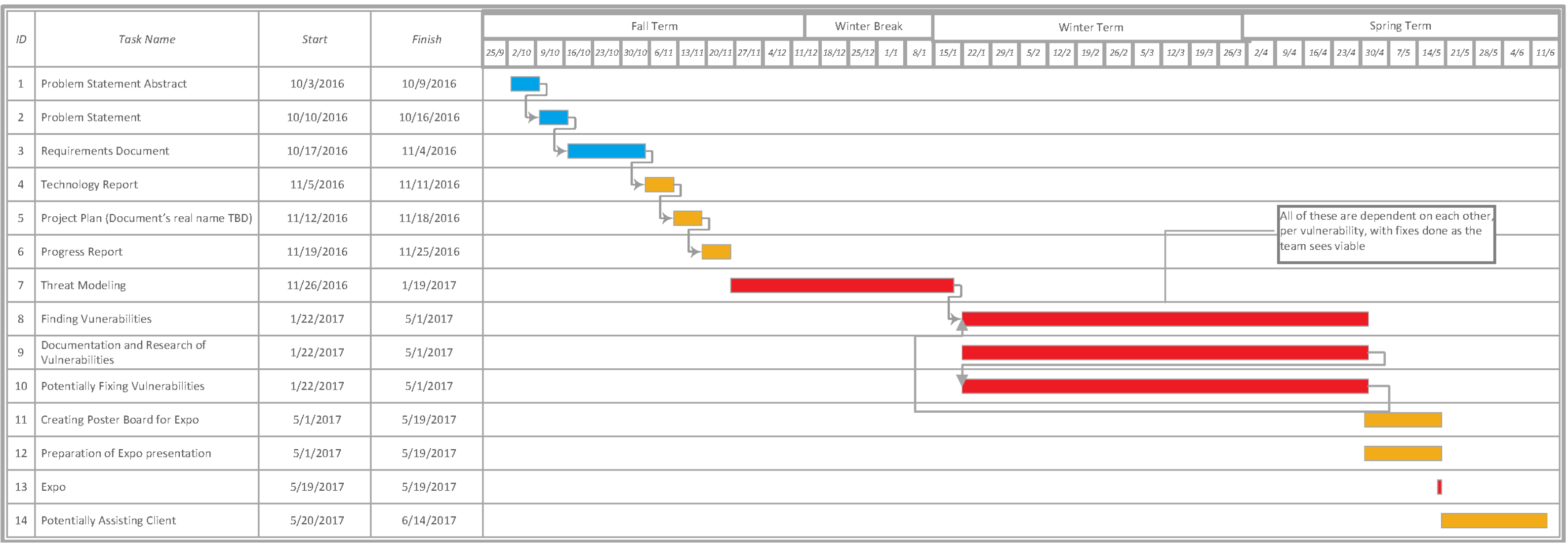
4.3 Constraints

Any work done on vulnerabilities needs to be appropriately documented and preferably published upstream to ROS/SROS as a whole.

4.4 Assumptions and Dependencies

There are only 3 real dependencies for this project at time of writing. First, we must complete our threat analysis before we start looking for threats, so that we have a general idea of where to investigate further. Second, we need to find a vulnerability before we can document it, or fix it, for obvious reasons. Lastly, we must have some sort of vulnerability at least documented before we prepare for Expo, or we won't have anything to present on.

4.5 Apportioning of Requirements



ID	*Task Name*	*Start*	*Finish*
1	Problem Statement Abstract	10/3/2016	10/9/2016
2	Problem Statement	10/10/2016	10/16/2016
3	Requirements Document	10/17/2016	11/4/2016
4	Technology Report	11/5/2016	11/11/2016
5	Project Plan (Document's real name TBD)	11/12/2016	11/18/2016
6	Progress Report	11/19/2016	11/25/2016
7	Threat Modeling	11/26/2016	1/19/2017
8	Finding Vunerabilities	1/22/2017	5/1/2017
9	Documentation and Research of Vulnerabilities	1/22/2017	5/1/2017
10	Potentially Fixing Vulnerabilities	1/22/2017	5/1/2017
11	Creating Poster Board for Expo	5/1/2017	5/19/2017
12	Preparation of Expo presentation	5/1/2017	5/19/2017
13	Expo	5/19/2017	5/19/2017
14	Potentially Assisting Client	5/20/2017	6/14/2017

All of these are dependent on each other, per vulnerability, with fixes done as the team sees viable

Figure 4.1: Gantt Chart

5 Specific Requirements

5.1 Software Interfaces

As we are looking for vulnerabilities in ROS, all code produced must be compatible with it or integrated into it. The version of ROS we will be using is Kinetic Kame, the current long-term support version of ROS.

5.1.1 Communications Protocol

ROS uses 2 major forms of communication. Internally, ROS has the publisher subscriber system, which works basically like a socket system. Publishers export data, and anyone who subscribes to that publisher receives the data, with no limit to the number of subscribers or any authentication on who can subscribe. There is also normally some sort of wireless/wired connection to a base station, which controls the starting and stopping of the robot. These generally take the form of a standard LAN connection, although with extra effort more complicated setups are possible.

5.2 External Interfaces

The external interface we will be working with is a DJI FlameWheel 550 HexCopter. Included on it is a Beaglebone Black with a PixHawk v1.6 Fire cape mounted on it. The PixHawk has a number of sensors, and interfaces directly with our 3DR GPS mounted unit. There is also a controller which communicates with the drone over 2.5 Ghz RF, along with a 900 Mhz MAVLink telemetry radio, that communicates with our ground station. The ground station is a computer running Mission Planner, a misison control software that allows us to define flight paths, and read telemetry data from the drone.

5.3 Functions

Any vulnerabilities found should in some way compromise the functionality or integrity of the robot. In turn, any fixes created for said vulnerabilities should prevent those from being compromised.

5.4 Specific Requirements

As previously stated, at the moment Group 50 is in the process of finding specific vulnerabilities. When we do find one, we will produce documentation outlining at least but not limited to: Our operating environment, the type of attack, the particular system/-piece of code attacked, the success rate of the attack, the result of the attack, and the potential fix to prevent the attack.

5.5 Software Attributes

5.5.1 Reliability

The reliability of ROS is what we will be trying to ensure, specifically that of it's security. We also want our documentation of any found exploits to be clear and readable for anyone using them in the future.

5.5.2 Availability

ROS and the potential exploits that we will be studying is open source, so it is available to anyone to confirm. If we are able to create a fix or patch for any of these we would contribute them to the open source SROS project.

5.5.3 Security

Security is the backbone of our entire project, we will be working specifically on finding potential security holes in ROS. If possible we hope to be able to create a patch for ones we are able to exploit and help to improve the security for everyone.

5.5.4 Maintainability

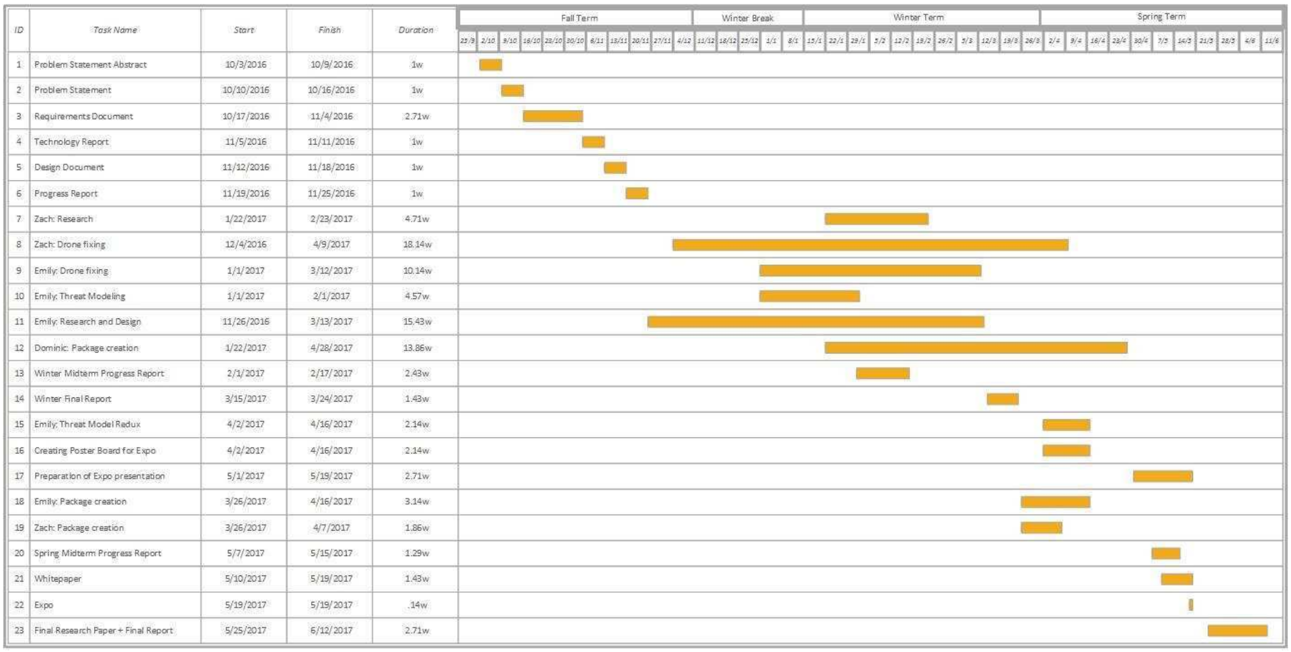
Because ROS is actively supported the software is maintained. If we produce a patch we would submit it to the SROS project and have it maintained within that.

5.5.5 Portability

ROS itself is very portable and we will be putting it onto our beaglebone. This also means that if we created a patch it would need to also be portable to the SROS project.

1 Changes to the requirements

1	Use Kinetic Kame as our ROS distribution	Used Indigo Igloo instead	The older version of ROS was more commonly used in other code samples, so the team used it instead for compatibility
2	Submit patches for vulnerabilites upstream	Didn't create any patches	All the vulnerabilities we found were based in how ROS was designed, not a code exploit or bug. So there really was not anything to patch.



ID	Task Name	Start	Finish	Duration
1	Problem Statement Abstract	10/3/2016	10/9/2016	1w
2	Problem Statement	10/10/2016	10/16/2016	1w
3	Requirements Document	10/17/2016	11/4/2016	2.71w
4	Technology Report	11/5/2016	11/11/2016	1w
5	Design Document	11/12/2016	11/18/2016	1w
6	Progress Report	11/19/2016	11/25/2016	1w
7	Zach: Research	1/22/2017	2/23/2017	4.71w
8	Zach: Drone fixing	12/4/2016	4/9/2017	18.14w
9	Emily: Drone fixing	1/1/2017	3/12/2017	10.14w
10	Emily: Threat Modeling	1/1/2017	2/1/2017	4.57w
11	Emily: Research and Design	11/26/2016	3/13/2017	15.43w
12	Dominic: Package creation	1/22/2017	4/28/2017	13.86w
13	Winter Midterm Progress Report	2/1/2017	2/17/2017	2.43w
14	Winter Final Report	3/15/2017	3/24/2017	1.43w
15	Emily: Threat Model Redux	4/2/2017	4/16/2017	2.14w
16	Creating Poster Board for Expo	4/2/2017	4/16/2017	2.14w
17	Preparation of Expo presentation	5/1/2017	5/19/2017	2.71w
18	Emily: Package creation	3/26/2017	4/16/2017	3.14w
19	Zach: Package creation	3/26/2017	4/7/2017	1.86w
20	Spring Midterm Progress Report	5/7/2017	5/15/2017	1.29w
21	Whitepaper	5/10/2017	5/19/2017	1.43w
22	Expo	5/19/2017	5/19/2017	.14w
23	Final Research Paper + Final Report	5/25/2017	6/12/2017	2.71w

Retrospective Gantt chart of tasks for Senior Capstone 2017 team # 50.

Security for Robotics

Oregon State University
College of Engineering

CS Capstone Design Document

June 13, 2017

Security for Robotics

Prepared for

Oregon State University

Vedanth Narayanan ______________________ ________
Signature *Date*

Prepared by

Group 50
RoboSec

Emily Longman ______________________ ________
Signature *Date*

Zach Rogers ______________________ ________
Signature *Date*

Dominic Giacoppe ______________________ ________
Signature *Date*

Abstract

In drones and other networked robotics there is a broad array of security vulnerabilities that can be leveraged in an attack. We will evaluate the ROS to find as many of these security holes as we can and document them. The different vulnerabilities found will be categorized into malware, sensor hacks, network and control channel attacks, and physical breaches. For some of these exploits we may be able to implement solutions, which will also be documented. These findings and any solutions will be added to an ongoing academic effort to make robotics more secure.

Contents

1 Introduction

1.1 Problem Definition

Robotics is still a relatively up and coming field and as most efforts in robotics are in pursuit of furthering capabilities, security has been left largely untouched. Because of this many functioning, deployed robots have limited to no security in their internal systems, making them very vulnerable to attacks. While the community developing robotics is still largely academic and there is little worry of being attacked, the security vulnerabilities still exist. Soon robotics will become ubiquitous in society and hackers will exploit these vulnerabilities for personal gain. There have already been reports of smart appliances being used for botnets, it's only a matter of time before drones and other robotics are similarly abused. [1] Through our work we hope to eliminate some of this abuse before it begins.

Since robotics are continually in development and there is such a wide variety of devices we have to focus our efforts onto a small subset of robotics to have any hope of making progress within a year. The Robotic Operating System (ROS) can be attacked at the driver level with malware, can have the configuration files modified, and can have data intercepted or spoofed in the internal communication subsystems. We should also investigate the ability to spoof sensor data, such as the camera, IR guidance, accelerometer, or gyroscopic systems. Outside of the OS level there is a huge amount of room for exploitation in the communication and control channels used by networked devices. Even physical attacks are something that needs to be addressed as a security concern. It's very important that we acknowledge all of these risks, since exploitation of them could be a big setback for global trust of robotics.

Essentially the problem we face is twofold; first we need to isolate a specific feature of a specific device, and then we need to attempt to break that feature, and document our work. This could be a crucial improvement for the use of robotics running ROS in a wide variety of sectors. The military uses and plans to use them widely, and for them more than anyone they need to be as secure as possible. Amazon and other commercial shipping companies have a vast use of robotics, and we've all heard about their plans for delivery drones, which are a huge security risk. If consumers think these will be abused they won't trust this upcoming technology and it will be slowly or never adopted, which will make companies hesitant to invest in their development. Consequentially, the world as a whole will be slower to advance robotic technology. Hopefully our work in securing robotics can help to better develop consumer trust in this emerging technology.

Emily

2 Research Design

2.1 Threat Models

A key element to our project is being able to accurately find vulnerabilities or identify areas where they are likely to be. One of the best ways to do this in an organized way that can be later referenced is with threat modeling. This method is widely used in industry when producing any sort of software. Many security professionals use it, but anyone can employ its methods. There are a variety of tools and technologies that can be used in threat modeling, which I am going to be comparing.

Adam Shostack mentions in his book on threat modeling that [2, p.203] One of the most important tools we can use is a whiteboard. This may not be a true technology but its important for us to be able to visualize the various modes of possible attack. It's perfect for initial creation of the models as we're sure to make many changes to ideas.

After using whiteboards to create our model visualizations though, tools like Visio can be extremely useful for creating a more formal, digital version. The Visio creations can also be easily shared and modified by anyone in the team, which makes distributed work easier. It also serves as a useful reference and record throughout the research process. If our models change at all we'll still have a formal record of any previous versions which we used to collect any previous data.

One last and important technology that Adam Shostack mentions is a bug tracking system. Just the issue tracking system in git could work for this, but there's a variety of ways we can do it. What's important is that it includes a definition of the threat, how we're going about it, the need to test it, the need to validate an assumption, and any possible mitigation we have. [2, p.205] We can then update this as we make progress on that specific exploit, and eventually close it as either completed or a dead end. To get the most out of threat modeling it would be great to employ all of these, but they may not all be viable.

Three threat models will be created for the purpose of this research, specifically one for the hardware components, one for the OS, and one for the communication channels. This allows the models to be more in depth as they can focus more within each specific area, providing a greater range of potential exploits. Each of these will be the roadmap for their sector and will be looked back on often in the iterations of researching each specific vulnerability. The research iterations themselves, the process by which each threat is investigated, provide a path for covering as much of the threat models as possible. Researchers will look at their threat model, choose the most promising starting place, examine the vulnerability as thoroughly as possible, and record all data found. They will then determine what degree of failure can be achieved and record that, then lastly try to find a way to mitigate any successful exploitation. This same process will be followed for all selected threats. Not every area from the threat model will be successful, some may not be possible to breach, at which point the cycle would be cut short on that specific piece and the researcher would move on to the next.

These threat models also serve a secondary purpose, of acting as a design viewpoint for those not versed in the technical side of the project. They will show how much potential there is in each area for exploitation, which helps someone looking at it to understand why that area is important to research and secure. It provides a visual representation of where the greatest concentrations of vulnerabilities lie, for both the researchers to work from and for anyone else to gain some understanding at a glance.

Emily

2.2 Documentation of Data

The core of the project is the data recorded when trying out different exploits on the system, and for it to be useful it must be recorded thoroughly and consistently. Because there is such a wide variety of methods which will be used to attack the system, the data will be quite varied. As much as it would be wonderful if it were possible to have easily comparable data from all of them, it simply isn't. To combat this what we come up with must be recorded consistently.

An article from Georgia Tech on proper research data documentation gives a few great concepts to focus on. [3] It states that among a number of things, one should record the location, methodology, software used, and any data processing done. For location data this means recording where the data was taken, which for our project might include GPS data when testing with the drones. It could also mean what location within the drone, and within that it could even be as specific as where in the code (if it's a software based exploit). Methodology is possibly the most important one, as it records how the data was generated, the protocol used, or where it came from. Our diverse range of data could actually be somewhat standardized if we meticulously record the methodology used and then later visualize it somehow according to that, most likely in some sort of tree based on categorizations of methodology. Software used ties into this, as it is part of what generates the data points we output. If software is used in manipulating, organizing, or visualizing the data, that would need to be recorded separately. This leads to the last point of data processing, which is also a very important one. If the data is changed in any way while processing it, that absolutely must be recorded, or else the legitimacy of the final conclusions could be ruined. Just like citing any sources used, anything done in the project must be cited for the sake of reproducibility.

While they are options, all of these should be taken into account when starting to actually produce data. The previously stated ideas don't necessarily need to be followed as they are, and there are sure to be problems with them or better options along the way. The main concept here is that research data is precious and sensitive and that integrity needs to be preserved to make our final product something that's academically valuable.

Emily

2.3 Data of Interest

There is wide variety of data to collect in this project, and that data is different for each class of attack. This makes drawing any overarching final conclusions difficult, as well as defining which exploits are the most severe. One form of data is universal though, and that's the failure mode of each successful attack. Failure Mode Effects Analysis (FMEA) is a procedure used in a variety of fields which creates an empirical system for testing and logging any failures. Somewhat akin to risk analysis, FMEA involves defining severity, occurrence, and detection rating scales, usually from 1 to 10. After these scales have been defined one can use them to calculate a risk priority number (RPN) and a criticality number with which the found failures can be mathematically ranked in order of importance. [4]

Employing this method and applying it to all successful exploits found will be immensely helpful in being able to compare completely separate systems. Final conclusion data can be drawn form these and an ultimate ranking of security prioritized vulnerabilities would be an immensely useful product to the community as a whole.

Emily

3 Approach

3.1 Methods of Data Capture

Capturing communication data between the user flying the drone, and the drone itself, will allow us to reverse engineer the communication protocols being used. Being able to capture that data also presents the possibility of doing a Man-In-The-Middle (MITM) like attack, enabling an attacker to intercept and spoof commands being sent to the drone in real time.

In order to capture this data, we need hardware that can receive RF on the 2.4Ghz and 900Mhz band. There are many, many options out there, some of which can be quite costly. Since the primary method of communication is through the 2.4 Ghz band, we can use a standard wireless radio that can run in promiscuous mode[5]. Promiscuous Mode enables us to capture wireless packets without associating with an access point. This is how a lot of wireless attacks are performed[5]. With this, we can use the Aircrack-NG suite of wireless auditing tools to attack the wireless communication channel that the drone uses[6].

While running in promiscuous mode a popular packet capturing tool known as Wireshark will also be helpful. Wireshark is a very powerful application that will allow deep packet inspection, which will aid in reverse engineering the communication channel[5].

With these tools we will be able to capture communication between the drone and the drone ground control station, allowing us to leverage that data to develop drone attack methods, relating to our communications threat model.

Zach

3.2 Drone Communication Channel

The two drones that we have use a 2.4Ghz data-link between the drone and the receiver ground-station unit. That receiver unit then uses a Bluetooth connection to connect to the user's controller, which is a physical controller or device such as a laptop or tablet.[7] With this in mind, there are two communication channels that can be targeted; the connection from the drone to the ground-station unit, and the connection from the ground-station unit to the controller. [7]

The 2.4Ghz frequency is commonly used by most Wireless Access Points, following the IEEE 802.11a/b/g standard. Bluetooth is another protocol that operates within the 2.4Ghz RF (radio frequency) spectrum[8]. This means that communication packets for the drone are being sent and received out in the open, which can be can be intercepted and analyzed with the right tools.

Each drone will be using a BeagleBone Black with a PixHawk Fire v1.6 Cape, making our drones Linux powered, running Ubuntu ARM, enabling us to use ROS[9]. The PixHawk is a flight control system, similar to the Naza-M2, which comes standard on the Flame Wheel ARF F550. It handles the drone's flight system, and includes a cluster of sensors (GPS, Gyroscope, etc) to keep track of vital information in order to maintain control over the drone while in flight. The PixHawk also handles telementry communication between the drone and the ground-station. As stated before, 2.4Ghz is the operating frequency between the drone and the ground-station, though the PixHawk also supports a RFD900 900Mhz Telemetry Radio, for longer telemetry reporting distances[10]. This opens up an additional communication channel that could be targeted. In order to intercept and analyze 900Mhz RF communications, we would need additional tools, seperate from what would be needed to intercept and analyze communications on the 2.4Ghz frequency[11].

What should now be clear is that there is a lot of data transmitted in the open air in order to have a successful drone system. This means that there are a lot of different ways that communications can be intercepted and even altered, in an attempt to gain control over a drone's flight plan. Knowing which communication channels to target is only a small part of getting to the ultimate goal of intercepting drone data; consider that the Research and Development phase. The next step is to explore how exactly to capture that data.

Zach

3.2.1 Network Setup

Wired communications shouldn't be any harder than inserting the Ethernet cable into the appropriate slots on the drone and the control station laptop. We may need to configure the Ubuntu install on the drone to recognize the Ethernet connection but it should be automated as part of the Ubuntu install. The wireless connection will be more involved, but should not be any different than a regular setup of wireless connectivity for Ubuntu based system. If the need arises many guides for setting up wireless connections on linux systems exist, like this. [12] Actually setting up the wireless card so that the board recognizes it and can use it is a different matter, and will be covered later on in this document. The control station wireless shouldn't be any different than the drone's, and may in fact be pre-configured for us as part of the initial installation.

Dominic

3.3 Drone OS

For the purposes of the OS, it is not so much its design as it is its execution, as most of the OS has been decided with the requirements of our pixhawk cape. We will be using ROS Kinetic, which is the recommended version of ROS for our setup on top of some board-specific software builds. The control station, AKA a desktop or laptop computer running linux with ROS for its own operating system, will have a trivial install and setup procedure. It is detailed here[13] but boils down to about 8 commands total and waiting for apt-get to finish. We foresee little trouble with the process. The more involved process will be flashing the drone with the correct software. To start, we will need to flash the BeagleBone black itself with a specific image found at this link along with instructions for the flashing process[14]. The general idea here is to flash the internal memory of the BBB with the system image so it has something to boot from. After that, we'll need to build the associated kernel with instructions on the same page and insert that into the beagle bone black so that it can operate the pixhawk. That is much the same process except we'll have to install some specific packages for cross compiling. Once that is complete, then we can build the pixhawk specific libraries for autopilot with instructions found on this page[15]. We would need to follow the advanced instructions at the bottom of the page, but they are no harder than command line git is and should not be too difficult for our team. Once this is all done, our drone will be using a version of arduPilot for our flight systems and a custom Ubunutu-based Linux build for the underlying operating system on the beagle board black.

Dominic

3.3.1 ROS v SROS (or rather ROS)

Once we have Linux installed on the station and Ubunutu on the drone, we can follow[13]. The gist of it is that it isn't much different than a regular desktop install; setup your sources, your keys, do some apt-gets, and ROS will be installed. The better question is what ROS packages we will install outside of the base ROS packages and the ones required for our board's autonomous routines for testing purposes. SROS is currently thought to be uncompatable with our drone's specific OS, as the autopilot packages on the drone use UDP based networking, while SROS only has TCP support for its encryption protocols. It is not known at this time if there is a workaround for this issue, or if SROS will just run the package in a ROS compatability mode, and as such the team will be using ROS for the foreseeable future.

Dominic

3.4 Leveraging Captured Data to Develop Attack Methods

Wireshark will also assist with analyzing the unknown communication protocol that the drone uses. Following the packet streams, and looking at the raw packet data, will allow us to form a concrete understanding of how the drone and ground control system associate with each other, and how commands are sent to the drone[16].

If reverse engineering proves to be unsuccessful, or difficult, we will still be in a poisiton to attack the drone communications, using MITM style attacks, and possibly some fuzzing related attacks[17].

Along with this analysis, monitoring how the drone ROS packages utilize the ROS Watchdog Timer will provide some further insight. The Watchdog Timer is used by ROS to detect system failures and crashes at both the hardware and software level[18]. Understanding the implementation of the watchdog and how it responds during various failures and attempts at exploitation will allow us to develop better, more sophisticated attack methods.

Zach

4 Conclusion

In the end, there are a few goals for the results of this project. First is just to gain insight into the penetrability of the drones current security and document it. Next is to determine where the biggest potential for risk lies and compile a ranking of the most potentially catastrophic vulnerabilities. Lastly the hope is to find or create as many patches as possible to prevent these exploits from being abused, and to contribute them to the community, possibly the SROS project.

By following the protocol that has been laid out and exploring as many of the options as possible, a great deal of valuable data can be generated. Future expansion upon it academically could yield even greater results, using it as a stepping stone for direction. The more research there is available on this subject the brighter the future for trusted robotics will be.

References

[1] S. Sharma, S. Garg, A. Karodiya, and H. Gupta, "Distributed denial of service attack."

[2] A. Shostack, *Threat Modeling, Designing for Security*. Indianapolis, Indiana: John Wiley and Sons, Inc, 2014.

[3] G. Tech. Document your data. [Online]. Available: http://d7.library.gatech.edu/research-data/documentation

[4] ASQ. Failure mode effects analysis. [Online]. Available: http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html

[5] Wireshark. Wlan ieee 802.11 capture setup. [Online]. Available: https://wiki.wireshark.org/CaptureSetup/WLAN

[6] A. NG. Getting started with aircrack-ng. [Online]. Available: https://www.aircrack-ng.org/doku.php?id=getting_started

[7] DJI. Naza-m2 flight control system. [Online]. Available: https://www.dji.com/naza-m-v2

[8] B. Barnett. Hacking at the 2.4ghz spectrum. [Online]. Available: http://www.grymoire.com/Security/Hardware.html#TOC

[9] BeagleBoard. Pixhawk fire cape: Linux drones with the beaglebone black. [Online]. Available: http://beagleboard.org/project/pxf/

[10] A. Project. Pixhawk overview. [Online]. Available: http://ardupilot.org/copter/docs/common-pixhawk-overview.html

[11] B. Barnett. Hacking the 900mhz spectrum. [Online]. Available: http://www.grymoire.com/Security/Hardware.html#Hacking_the_.3C1Ghz_Range_.28900Mhz_.29_Spectrum

[12] Linux.com. How to configure wireless on any linux desktop. [Online]. Available: https://www.linux.com/learn/how-configure-wireless-any-linux-desktop

[13] R. Project. Installing ros indigo. [Online]. Available: http://wiki.ros.org/indigo/Installation/

[14] A. Project. Building ardupilot for beagle bone black. [Online]. Available: http://ardupilot.org/dev/docs/building-for-beaglebone-black-on-linux.html#

[15] ——. Building ardupilot for pixhawk. [Online]. Available: http://ardupilot.org/dev/docs/building-px4-for-linux-with-make.html

[16] Polynomial. How do you analyze an unknown network protocol. [Online]. Available: https://security.stackexchange.com/questions/17344/how-do-you-analyze-an-unknown-network-protocol

[17] H. Böck. Network fuzzing with american fuzzy lop. [Online]. Available: https://blog.fuzzing-project.org/27-Network-fuzzing-with-american-fuzzy-lop.html

[18] Ros watchdog timer - detecting crashes. [Online]. Available: http://wiki.ros.org/watchdog_timer

Changes to the project design

We had 2 major changes from our original design: we did not end up using the drone at all, and we determined that SROS was not viable for a few reasons. The drone simply took too much of our time to get it working to be usable for the project. The team initially thought making the drone usable for our purposes would take about a week, by the time we decided to drop it from the project it was almost the end of winter term. (ZACH, fill in your drone struggles here, only like 2 sentences though save the big bash for the failures bit). SROS was dropped much earlier on in the project when the team discovered that the prefered installation method for it was containers, which the team had no experience with and might not be feasible on the drone, and that it was so poorly documented that the team did not feel that it was robust enough to be a working alternative to ROS. Overall, SROS was not determined to be worth the amount of extra overhead to get it operational for our purposes, and was subsequently cut from the project.

Security for Robotics - Tech Review

Emily Longman, Zach Rogers, and Dominic Giacoppe

CS461 Capstone

11/28/2016

Emily

Threat Modeling

A key element to the project is being able to accurately find vulnerabilities or identify areas where they are likely to be. One of the best ways to do this in an organized way that can be later referenced is with threat modeling. This method is widely used in industry when producing any sort of software. Many security professionals use it, but anyone can employ its methods. There are a variety of tools and technologies that can be used in threat modeling, which are compared below.

Adam Shostack mentions in his book on threat modeling that [1, p.203] One of the most important tools we can use is a whiteboard. This may not be a true technology but its important to be able to visualize the various modes of possible attack. It's perfect for initial creation of the models as they're sure to go through many changes.

After using whiteboards to create model visualizations though, tools like Visio can be extremely useful for creating a more formal, digital version. The Visio creations can also be easily shared and modified by anyone in the team, which makes distributed work easier. It also serves as a useful reference and record throughout the research process. If the models change at all there will still be a formal record of any previous versions which we used to collect any previous data.

One last and important technology that Adam Shostack mentions is a bug tracking system. Just the issue tracking system in git could work for this, but there's a variety of ways it can be done. What's important is that it includes a definition of the threat, how the team is going about it, the need to test it, the need to validate an assumption, and any possible mitigation found. [1, p.205] It can then be updated as progress is made on that specific exploit, and eventually closed as either completed or a dead end. To get the most out of threat modeling it would be great to employ all of these, but they may not all be viable.

Documentation of Data

The core of the project is the data recorded when trying out different exploits on the system, and for it to be useful it must be recorded thoroughly and consistently. Because there is such a wide variety of methods which will be used to attack the system, the data will be quite varied. As much as it would be wonderful if it were possible to have easily comparable data from all of them, it simply isn't. To combat this what we come up with must be recorded consistently.

An article from Georgia Tech on proper research data documentation gives a few great concepts to focus on. [2] It states that among a number of things, one should record the location, methodology, software used, and any data processing done. For location data this means recording where the data was taken, which for our project might include GPS data when testing with the drones. It could also mean what location within the drone, and within that it could even be as specific as where in the code (if it's a software based exploit). Methodology is possibly the most important one, as it records how the data was generated, the protocol used, or where it came from. Our diverse range of data could actually be somewhat standardized if we meticulously record the methodology used and then later visualize it somehow according to that, most likely in some sort of tree based on categorizations of methodology. Software used ties into this, as it is part of what generates the data points we output. If software is used in manipulating, organizing, or visualizing the data, that would need to be recorded separately. This leads to the last point of data processing, which is also a very important one. If the data is changed in any way while processing it, that absolutely must be recorded, or else the legitimacy of the final conclusions could be ruined. Just like citing any sources used, anything done in the project must be cited for the sake of reproducibility.

While they are options, all of these should be taken into account when starting to actually produce data. The previously stated ideas don't necessarily need to be followed as they are, and there are sure to be problems with them or better options along the way. The main concept here is that research data is precious and sensitive and that integrity needs to be preserved to make our final product something that's academically valuable.

Data of Interest

There is wide variety of data to collect in this project, and that data is different for each class of attack. This makes drawing any overarching final conclusions difficult, as well as defining which exploits are the most severe. One form of data is universal though, and that's the failure mode of each successful attack. Failure Mode Effects Analysis (FMEA) is a procedure used in a variety of fields which creates an empirical system for testing and logging any failures. Somewhat akin to risk analysis, FMEA involves defining severity, occurrence, and detection rating scales, usually from 1 to 10. After these scales have been defined one can use them to calculate a risk priority number

(RPN) and a criticality number with which the found failures can be mathematically ranked in order of importance. [3]

Employing this method and applying it to all successful exploits found will be immensely helpful in being able to compare completely separate systems. Final conclusion data can be drawn form these and an ultimate ranking of security prioritized vulnerabilities would be an immensely useful product to the community as a whole.

References

[1] A. Shostack, *Threat Modeling, Designing for Security.* Indianapolis, Indiana: John Wiley and Sons, Inc, 2014.

[2] G. Tech. Document your data. [Online]. Available: http://d7.library.gatech.edu/research-data/documentation

[3] ASQ. Failure mode effects analysis. [Online]. Available: http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html

Technology Review - Security for Robotics

Zach Rogers

CS461 Capstone

16 Feb 2017

Abstract

Our goal as a group is to identify vulnerabilities, both hardware and software related, within our drone system. A big part of that will have to do with the drone's communication channel, which describes how a user controls a drone during flight and general operation. In order to attack the communication channel, we must first understand how the drones communicate with the user, and how the user sends commands to the drone. This will involve lots of data capturing. So my focus right now is to determine how we will be capturing that data, and how we will use that data to reverse-engineer the drone's methods of communication for the purpose of developing attack methods.

Drone Communication Channel

The two drones that we have use a 2.4Ghz data-link between the drone and the receiver ground-station unit. That receiver unit then uses a Bluetooth connection to connect to the user's controller, which is a physical controller or device such as a laptop or tablet.[1] With this in mind, there are two communication channels that can be targeted; the connection from the drone to the ground-station unit, and the connection from the ground-station unit to the controller, this can be seen on Figure 1[1].

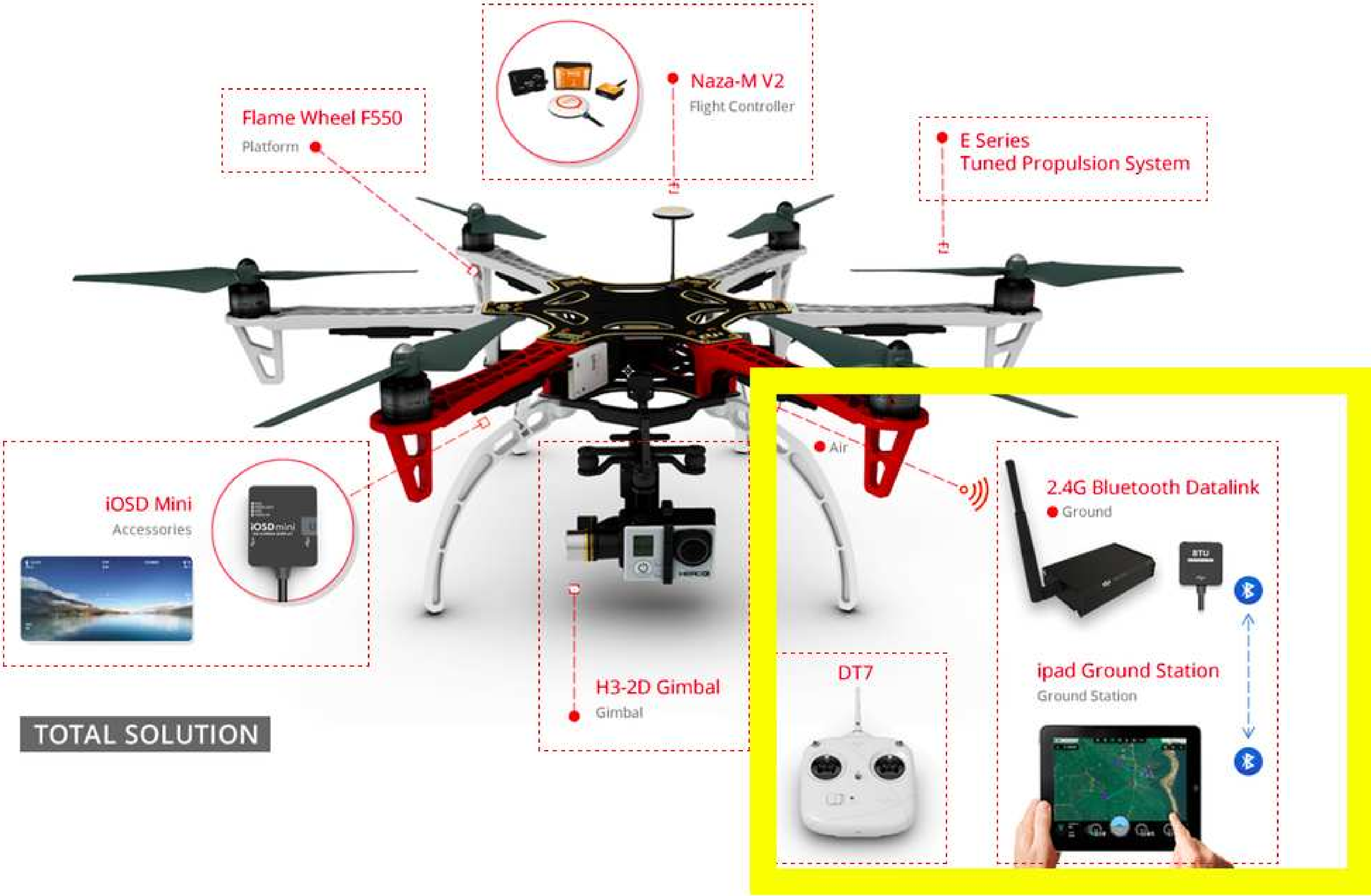


Figure 1: Flame Wheel ARF F550 Datalink Diagram

The 2.4Ghz frequency is commonly used by most Wireless Access Points, following the IEEE 802.11a/b/g standard. Bluetooth is another protocol that operates within the 2.4Ghz RF (radio frequency) spectrum[2]. This means that communication packets for the drone are being sent and received out in the open, which can be can be intercepted and analyzed with the right tools.

Each drone will be using a BeagleBone Black with a PixHawk Fire v1.6 Cape, making our drones Linux powered, running Ubuntu ARM, enabling us to use ROS[3]. The PixHawk is a flight control system, similar to the Naza-M2, which comes standard on the Flame Wheel ARF F550. It handles the drone's flight system, and includes a cluster of sensors (GPS, Gyroscope, etc) to keep track of vital information in order to maintain control over the drone while in flight. The PixHawk also handles telementry communication between the drone and the ground-station. As stated before, 2.4Ghz is the operating frequency between the drone and the ground-station, though the PixHawk also supports a RFD900 900Mhz Telemetry Radio, for longer telemetry reporting distances[4]. This opens up an additional communication channel that could be targeted. In order to intercept and analyze 900Mhz RF communications, we would need additional tools, seperate from what would be needed to intercept and analyze communications on the 2.4Ghz frequency[5].

What should now be clear is that there is a lot of data transmitted in the open air in order to have a successful drone system. This means that there are a lot of different ways that communications can be intercepted and even altered, in an attempt to gain control over a drone's flight plan. Knowing which communication channels to target is only a small part of getting to the ultimate goal of intercepting drone data; consider that the Research and Development phase. The next step is to explore how exactly to capture that data.

Methods of Data Capture

Capturing communication data between the user flying the drone, and the drone itself, will allow us to reverse engineer the communication protocols being used. Being able to capture that data also presents the possibility of doing a Man-In-The-Middle (MITM) like attack, enabling an attacker to intercept and spoof commands being sent to the drone in real time.

In order to capture this data, we need hardware that can recieve RF on the 2.4Ghz and 900Mhz band. There are many, many options out there, some of which can be quite costly. Since the primary method of communication is through the 2.4 Ghz band, we can use a standard wireless radio that can run in permiscuous mode[6]. Permiscuous Mode enables us to capture wireless packets without associating with an access point. This is how a lot of wireless attacks are performed[6]. With this, we can use the Aircrack-NG suite of wireless auditing tools to attack the wireless communication channel that the drone uses[7].

While running in permiscuous mode a popular packet capturing tool known as Wireshark will also be helpful. Wireshark is a very powerful application that will allow deep packet inspection, which will aid in reverse engineering the communication channel[6].

With these tools we will be able to capture communication between the drone and the drone ground control station, allowing us to leverage that data to develop drone attack methods, relating to our communications threat model.

It is also important to note that we need to get written approval from Oregon State University to intercept wireless traffic. During the time we intercept traffic, we will also capture legitmate traffic in the area that we are operating. We have spoken with OSU's Cheif Information Security Officer, Dave Nevin, regarding this approval.

Leveraging Captured Data to Develop Attack Methods

Wireshark will also assist with analyzing the unknown communication protocol that the drone uses. Following the packet streams, and looking at the raw packet data, will allow us to form a concrete understanding of how the drone and ground control system associate with each other, and how commands are sent to the drone[8].

If reverse engineering proves to be unsuccessful, or difficult, we will still be in a poisiton to attack the drone communications, using MITM style attacks, and possibly some fuzzing related attacks[9].

References

[1] DJI. Naza-m2 flight control system. [Online]. Available: https://www.dji.com/naza-m-v2

[2] B. Barnett. Hacking at the 2.4ghz spectrum. [Online]. Available: http://www.grymoire.com/Security/Hardware.html#TOC

[3] BeagleBoard. Pixhawk fire cape: Linux drones with the beaglebone black. [Online]. Available: http://beagleboard.org/project/pxf/

[4] A. Project. Pixhawk overview. [Online]. Available: http://ardupilot.org/copter/docs/common-pixhawk-overview.html

[5] B. Barnett. Hacking the 900mhz spectrum. [Online]. Available: http://www.grymoire.com/Security/Hardware.html#Hacking_the_.3C1Ghz_Range_.28900Mhz_.29_Spectrum

[6] Wireshark. Wlan ieee 802.11 capture setup. [Online]. Available: https://wiki.wireshark.org/CaptureSetup/WLAN

[7] A. NG. Getting started with aircrack-ng. [Online]. Available: https://www.aircrack-ng.org/doku.php?id=getting_started

[8] Polynomial. How do you analyze an unknown network protocol. [Online]. Available: https://security.stackexchange.com/questions/17344/how-do-you-analyze-an-unknown-network-protocol

[9] H. Bck. Network fuzzing with american fuzzy lop. [Online]. Available: https://blog.fuzzing-project.org/27-Network-fuzzing-with-american-fuzzy-lop.html

Security for Robotics

Emily Longman, Zach Rogers, and Dominic Giacoppe

CS461 Capstone

11/13/2016

Abstract

Mission My specific work for the project is, at the moment, managing the software components of the drone itself. Most of our attempts to find vulnerabilities will be at the software level, so deciding what exact software we are using is crucial in how we approach our research.

Drone Operating System

The most important component of the drone itself is the operating system, and as I see it our team has 3 major options: an embedded system, a real time operating system, and a non-realtime operating system. For the purposes of our design, the we will consider Nuttx https://bitbucket.org/nuttx/nuttx/src/master/ for our real time operating system, and Ubuntu+ROS http://www.ros.org/ for our non-realtime. The embedded system would have to be handmade by our team, so there isn't an existing one to use as a comparison. Indeed, this is the first major downside for an embedded system: not only would our team have to create one, it would be specific to whatever we are using hardware-wise and not be portable to other drones unless we made it so. The portability is in direct conflict with the overall mission of the project, which is to make robotics software more secure in general. If our fixes only work for one OS, run by exactly one robot, we aren't doing much to better drone security as a whole. As such there is no particular use for an embedded system for this project. It seems a little like a software equivalent of digging a hole just to fill it. As for nuttx, it avoids that problem by being one of the most used real time drone operating systems. I could not find any statistics on that, nobody seems to take polls on this subject yet, but based on how widely it has been ported and that it happens to be the first open-source result on google when you look up real time drone OS I can only assume it is at least popular. That also means that we will likely have a pre-existing port available for whatever board we decide to use, and will not need to do the porting ourselves, which saves us the possibility of introducing security vulnerabilities ourselves. We would still however have to configure and build nuttx itself on the drone, which may or may not be a problem based on the readme, along with making sure our power supply is compatible. And, as every board's version of nuttx is board specific, any fixes we create are only good for that board's version of nuttx. It does have it's benefits: if setup correctly the team can squeeze out as much power as possible from the drone, and that real time operating systems seem to be the way most commercial and military drones are heading, which means our research is more applicable overall. However, ROS+Ubuntu have the distinct advantage in 3 areas: our project was originally written to use ROS, that ROS itself is not build-specific and is in fact portable to run on top of nuttx among other things, and that 2 members of our group (Zach and I) both have prior experience using ROS. ROS also happens to be a very robust project at this point with a large software ecosystem, and our campus has a ROS expert in Prof. Bill Smart who we could tap for insight. Lastly, while the drone community is moving towards real time operating systems, for robotics as a whole that does not seem to be necessarily true, and while we will most likely use a drone for the project the goal overall is to improve security for robotics. ROS has a large market share on robotics, and at the moment seems to have little competition in the generic robotics system category. As such, our team will use ROS for our project.

ROS v SROS

SROS http://wiki.ros.org/SROS is a port of ROS with the specific focus of being a more secure version of ROS; in essence it aims to achieve a more complete version of our team's goal with a ROS focus. We both desire to better the security of robotics, in particular ROS, through adding security minded enhancements and patching potential vulnerabilities inherit to the design of ROS. It also already has some features that our team would have considered adding to ROS anyways, like TLS support for ROS sockets and certificate management. The project also has a small roadmap for it's goals, which we would give us targets for us to investigate. Lastly, it has an install guide for ubuntu systems, so our team shouldn't have too many issues installing SROS compared to regular ROS. It has 3 major drawbacks, however, as compared to focusing on regular ROS. The most major one is that the SROS project, compared to ROS, is very young with very little general support or package support. Also, based on the last edits to SROS's wiki pages, there has been no work on the project since August, all work up until this point has been done by 2 users, and it's wiki pages are covered in TODOs. In fact, the entire documentation for this project comprises a grand total of 10 pages, none with real explicit details on the current differences between ROS and SROS. In addition, this is the official github repo for the SROS project: https://github.com/osrf/sros. It is almost completely empty, with most page's last commits dating back to June of this 2016. This suggests the project has been closeted by it's creators, which in turn suggests that either they lost interest in the project or that was simply not feasible as planned. Neither of those are good signs for our project, and would suggest that to work on SROS is to take control of the project from it's creators. The next issue is that since it is such a small project, the user base is much smaller than ROS'es, if it indeed has one. Any work our team accomplishes would need to be incorporated into the main ROS distribution channel for it to see much use, and hence we are improving robotics security for a much smaller subset of people. For SROS to be fully adopted by the ROS community, it would need to become a fully fledged ROS equivalent with package support, which is rather outside the scope of our project. We are aiming to improve security, not necessarily make ROS completely secure within 4 months. Third, a lot of SROS'es improvements are tied up in the use of containers, which while fine on it's own limits it's usefulness to systems that can support the amount of overhead a container creates. For bare-metal systems this may not be possible and thus limit our potential audience. Also, nobody on our team has any experience with containers, which would be a limiting factor for our team. For the reasons above, unless the SROS project shows signs of life anytime soon, we will be using basic ROS for our project.

Performance Metrics

wireless/wired/no comms A major factor in how we approach our research is what kind of connection our team will have between the drone and the control station, the control station being the computer the user controls the drone from. We have 3 options: a wired connection, probably some sort of Ethernet cable, a wireless connection using an ad-hoc network between the control station and the drone, and forgoing the control station entirely in favor of a fully autonomous drone. There is also the possibility of using both wired and wireless connections at different points in the project, but a fully autonomous drone would require a very specific drone setup that would be difficult to revert to and from during the course of our project and hence will be considered separately. The major benefit of a wired connection would be that it would have the least possible connection problems and the highest connection latency. The downside is the our current probable drone model flies using propellers, which means that cables could have the possibility of getting caught in them. It also limits the distance we can fly the drone, and means that we have to maintain a line of sight between our control station and the drone at all times, lest we break the cord or disconnect the drone from the control station which would probably result in a crash. Also, while a wired connection offers the most stable and fast connection possible, for the purposes of our project connection speed will not be a major factor in our results. While it would be nice, we are not trying to do anything with the drone per-say and have no intention of doing any operations where milliseconds are the difference between success and failure. What a wired connection would be especially good for in this project would be bootstrapping and flashing the drone as we attempt to modify the drone's OS. Wireless connections require more involved drivers traditionally than wired ones, and the less bootstrapping we have to do lessens the probability of user error. Wireless, however, allows us to have more range on where we can fly the drone, along with allowing the drone controller station to remain completely stationary, and eliminates any concerns about tangling or catching the cord on random objects. However, the project also doesn't really need a large range of flight on the drone, and probably won't go more than a foot or two off the ground for our purposes, so the extra range may not be necessary. Wireless has some potential to drop packets and have signal delays however, which may result the drone doing things it wasn't supposed to do stall in midair, or crash. It does. Also, for both wired and wireless connections, we will have to setup some sort of listener protocol on the drone to allow for communications. Assuming we use ROS this is already made for us, but we will still have to configure our control station to communicate properly with the drone. Lastly, a fully autonomous drone has the distinct benefit that we would not need to create a controller station and pilot the drone at all, which frees up another member from piloting duty. The 2 major downsides are that we would need to create our own autonomous routine, which is a lot of extra work up front for us to produce that is limited to our drone in particular and may not reflect the codebase for autonomous drones as a whole. Additionally, most drones in use today are not fully autonomous but require a human pilot, which means our work may or not be as applicable to drone security as a whole. The only fully autonomous drones I can find in use currently tend to be military drones that perform reconnaissance missions and return, mostly UAV drones. As most of these drones are the products of DARPA and other state agencies, any advances we make in drone security will probably have little affect on autonomous drone security as a whole, unless DARPA decides to start using ROS. As such, we will probably use a combination of wired and wireless communication systems with a control station. Wired when we need to update drone software, and wireless for piloting the drone.

Technology changes

The only technologies that were cut were those related to the drone, and SROS. Since we had no drone, we had no need for a drone operating system, nor did we care about drone networking or communications, and hence those were all ignored. The reasons for dropping SROS are detailed as part of the design document discussion, and the team used regular ROS for the rest of the project. Our project did not use any additional technologies not listed in the tech doc.

1 Dominic's Blog Posts

1.1 Weeks 0-3, Fall

This is the first blog post. This last week was mostly continuing to hash out details with our client, and figuring out what direction we want to take on the project. Next week should be more or less the same; although we will get to see our probable test drone next week. Only problem this week was figuring out when and where to meet

1.2 Week 4, Fall

This week was mostly about clarification and finding out that we aren't doing our paperwork correctly; also we got to see the drone. The drone runs beaglebone-black stapled to a pixboard, and should run ROS. Also it has 6 propellers, a programmable power supply, and a honking battery. As far as goals go, we are more or less set on finding/fixing vulnerabilities in ROS at this point, as it has nice known big ones. It sounds like we might work with the SROS (secure ROS) project, but the SROS project is also in it's infancy and only has some goals on it's page. Also, we met Jesse, an EX Intel employee with heavy crypto/security experience, and I look forward to learning under him.

1.3 Week 5, Fall

Most things are temporarily on hold while midterms come. We did finish our draft for our project specification, however. Probably needs revision but we'll wait for the feedback. Still need a team name

1.4 Week 6, Fall

Midterms have finally passed us, my grades remain in a solid B range. Life goes on. As far as the project goes, we will be meeting tonight to put the finisher on the project specifications, sign it, and turn it in. We also got access to our drones this week. We will be in charge of fixing one of them (prior student broke something), and adding the parts we will be using (beaglebone black, some legs) before winter.

1.5 Week 7, Fall

Not much this week. Mostly attempting to make progress on the Tech doc, but everyone but me seems to be busy. Other classes have more for me to do, so...

1.6 Week 8, Fall

This week was trying to get a tech doc together. Turns out we were all sick and busy except me, which explains why nothing got done. This means we are spilling over into design doc time, but that has it's own issues. Still not sure what to write about really. Also we are using a pixhawk, not a beaglebone?

1.7 Week 9, Fall

Class is basically over tomorrow so I might as well write now. We met today to discuss how we are going to finish the tech doc, along with further debating some technical aspects of the project, most specifically for me whether or not we will be using SROS or not, and if our board can support the latest version of ROS. We also discussed the preliminary aspects of the design doc, which I will start on about now.

1.8 Week 10, Fall

We got the design doc in on time. Wew. Now on to the final boss(es): My 3 finals, and putting together the progress report and video. Hopefully that happens on time. EDIT: It all happened on time. Did well on my finals too!

1.9 Week 1, Winter

We are back to work. Very little happened over break due to the team being scattered on the four winds, but we have now met up and discussed our current plans. We aim to have the drones functioning by the end of next week, along with completing the threat model.

1.10 Week 2, Winter

Progress was made with getting the drones operational, and in theory the threat model is done. However, the drones have some board issues(something about busted connections) and it'll take some more ECE magic before those work. Also, while we have setup the Beaglebones with our linux/ROS install, it's having some memory read issues and it doesn't always boot correctly at this time; we are looking into this but it may be a hardware issue with our SD card reader. In good news I've started working with Vee to create a malicious ROS package for testing purposes, we have one most of the way there but are running into some compile dependency issues about graphics drivers. I'm also making one separately that will be a fair bit less complex and should have it done in the near future.

1.11 Week 3, Winter

I finally wrote code this week. I made some script-kiddy-esc ROS packages that do simple bad things like fork-bomb and kill system stuff. All of these require that you have the bad packages on the system already, but that's okay. Working on thinking up more unique bad packages to make at this time.

1.12 Week 4, Winter

Not a ton happened this week. We didn't meet as a team Tuesday because everyone was busy, but then Thursday we found out we have more documentation to do, so we are re-gearing for completing that. Emily is our captain by vote, and expo registration should be done soon

1.13 Week 5, Winter

More progress this week. I made another package, but we were also told of a new high directive: rewriting our documentation to fit current needs. That will happen nowish.

1.14 Week 6, Winter

The documentation rewrite overrode all other priorities, but it got done. We are looking to make further progress starting next week.

1.15 Week 7, Winter

The group has determined that there were some internal problems, and are working on refocusing our efforts. Specifically we are looking into whether making the drone work is feasible anymore, and exactly how we are structuring our research.

1.16 Week 8, Winter

Our issues have been more or less resolved. The proper authorities have been spoken to, we got help for figuring out what was wrong with the drone, and progress is being made again. And we got some clarifications on research specs, but mostly the drone. Next week is the show up or die class, so we will be showing up.

1.17 Week 9, Winter

Progress continues to be made on the drone, and continuing to bug McGrath for help seems to be working for when we get stuck. However progress is temporarily halted until we get another batch of documentation over with, mostly the poster. This, along with a different poster for a baccore class I'm taking, means I have 2 art projects due by next Friday. And I am not an art person.

1.18 Week 10, Winter

We met briefly this week and worked on the drone some, but most of our attention was now on our progress reports. Mine is done, but then I had other classes. I thought of an idea for my next malpac at least. Also, something broke my markdown and this is now a big chunk of text. It doesn't seem to want to fix itself, either...

1.19 Week 1, Spring

So I need to use this 3 P format thing.
Progress: Over break I made a bunch of new packages as I hoped I would. About 7 total.
Plans: Make more packages, prepare for expo. Make sure the group is meeting with the right people. Check to see somethings about ROS message rates.
Problems: Still no drone, possibly won't even be able to do as much as we like with it because we won't have security clearance if the situation doesn't change shortly.

1.20 Week 2, Spring

Progress: We met with Vee Friday, and after some discussion the rest of the team is now fully aware that they need to make code, hopefully soon. Also we are deep 6'ing the drone at this point.
Plans: At this point I think I stop making my own packages for a bit and make sure the rest of the team gets off the ground in short order.
Problems: 3 weeks until expo and we need more code badly

1.21 Week 3, Spring

Progress: We met with Vee again Thursday, and he's happier with our progress. I gave some ideas to Emily to code but Zach found some stuff on his own. I also have a new idea to work on but I doubt I can really make it happen...
Plans: Work on my own package, support Emily as needed. Work on poster?
Problems: Hand encoding TCP packets is really not fun.

1.22 Week 4, Spring

Progress: Poster is near done. We just need to take a group pic. And I worked on that package from last week, but I have determined it's just not feasible. Also fixed this page!
Plans: Think about packages, if I can come up with anything else try for it
Problems: My other classes are hellish and I'm mentally tapped.

1.23 Week 5, Spring

Progress: Poster got done. I still don't have anything good for new packages though....
Plan: Probably think about midterm progress report for now.....
Problems: Why does the homework never end

1.24 Week 6, Spring

Progress: We have begun work on the midterm progress report and the final paper. No packages though, not until after expo
Plan: Work on papers, try not to fail other classes
Problems: so much due next week

1.25 Week 7-10+Expo, Spring

EXPO Progress: We did our paperwork and Expo is over Plan: Finish this research paper thing, maybe another package or two
Problems: None at the moment?
If I got a redo, first thing to do is just scrap the drone from day 1, or get the ISO from McGrath sooner.
Also, maybe aim to make package 15 my big final package and actually finish it. But mainly, make sure to take ECE 375 in winter term, meet with Vee every week and do a better job as relay.

Biggest skill is team communication bar none, including breaking bad news.
I have no idea. Does respiration count?
How to adapt to change, rapidly. Also disaster planning and management.
No, not really. I don't feel like our code base has enough variance in attack types to be worth much.
More packages with more variety. Maybe actually fix the drone.

2 Emily's Blog Posts

2.1 Weeks 0-3, Fall

The first few weeks of this course were spent analyzing which projects we wanted, who we would be working with, and how we wanted to initially approach our given problem. It was somewhat of an adjustment period to the course and the next nine months. The calm before the storm essentially. These weeks also provided us with an opportunity to prepare for all the technical writing and research we were about to do.

2.2 Week 4, Fall

During this week we continued our work on getting the project better defined and potentially revising our problem statement. We also got to have a great meeting with Rakesh and Jesse, who gave us some really cool professional insight into what we could potentially focus on. We've mostly confirmed that we're using ROS at this point, and we have some ideas about ways to attack it, but nothing has really been solidified yet. We do hope to be able to contribute to the SROS project, but we're not sure it will be usable for us.

2.3 Week 5, Fall

This week was somewhat of a lull, but we were able to get our problem statement revised, which will help us. We also started work on an initial draft of our requirements document in preparation for the final version due next week. We met as a group to discuss it and have a great plan for moving forward. We also now have seen our drones, but are lacking some components still.

2.4 Week 6, Fall

Despite our solid first draft we still scrambled to pull together our final version of the requirements document, mainly because it was a little bit unnatural for a research project. There was still so much ambiguity that it seemed like we were pulling at straws and making wild guesses as to what we need. As for the hardware we do now actually have our drones but we need to do some surgery on them and I need to resolder a component on one of the PixHawk boards that Sam broke.

2.5 Week 7, Fall

This week we made some rough sections for the tech review, but it's been really difficult for us to figure out enough of them that make sense to do a tech review on. The majority of our project is based around the research lifestyle and ROS, but outside of that we don't have a ton to talk about. Hopefully when we have a more specific view of what we're doing we can come back and better define this.

2.6 Week 8, Fall

Things were a bit of a struggle this week and not a lot got done. I've had a cold and Zach has also been sick, so it was mostly just each of us trying to finish our tech doc sections. A lot of other classes have gotten chaotic this week too, so we were definitely not as focused as we needed to be on this.

2.7 Week 9, Fall

This week was a short one with the holiday, and was spent trying to catch up on the tech doc that both Zach and I failed to complete by the deadline. I've talked with both Kirsten and Kevin about it and they had useful pointers, but it still seems like such a stretch when writing it. We're also starting to think about the design doc coming up next week and the end of term assignments.

2.8 Week 10, Fall

This was the last week of classes and we focused quite a lot of our effort into making the best design document we could. I'm actually really happy with the outcome, and it was probably the most useful of the documents we've written this term. I do wish that I'd had just a little more time to add a better literature review section per Kirsten's suggestion, but we can still add those sources into our final progress document. That's all we have left to do, other than make the video presentation of it. I can't believe this term is already over.

2.9 Week 1, Winter

Over break we each individually did some work on the project. The portion I worked on was creating our threat models, which we need to use to complete our research iterations this term. This week I'm adding in some of the sources that Dominic and Zach found over break to finish fleshing them out and finalizing them. We are all also going to get the hardware fully functional this week so that we can begin testing as soon as possible. We've met as a group to discuss our project but still need to meet with Vee and our TA.

2.10 Week 2, Winter

This week we made headway with the hardware, getting the pieces we aren't using taken off of the drones and Zach getting ROS installed on both the boards so that they're ready to go. I get the threat models finished up and published, but there's discussion of changing their layout so they may get more tuning this coming week. In finishing them I've amassed more pages and articles we can use, so we have even more to go off of. We also got to meet with Jon this week so we have a better idea of what's ahead of us this term, as well as get his input on where we are in the project. I think we're all hoping to get a lot done this coming week.

2.11 Week 3, Winter

A lot got done this week, just as we had hoped it would. We met up three or four times and worked together on getting the boards set up with the operating system, getting some malware packages set up for testing, and getting the hardware ready to go. My part specifically was the hardware, which got everything returned to Kevin that we weren't using, a battery for one of the drones, and everything put in its place. We just need to hook up the boards now and they should be working. We're just making sure that they'll interface correctly with the PixHawks first, since that's been giving us a bit of trouble. In the mean time though we all have an installation of ROS on our computers so we can do some research and testing without having to use the physical drones. My first research prospect is sensor spoofing so I'll be doing more research into that this week.

2.12 Week 4, Winter

Week four was somehow yet another productive one. While I personally spent the majority of it sick in bed, I was still able to work on some sensor spoofing ideas. I researched the topic more thoroughly and was able to find some pretty great sources. The other two also continued to get work done on their respective research sections. We also had class for the first time in a while, which gave us some good pointers for what to expect in the next couple weeks, and how we should prepare for them. Early next week we'll be working on the rewrite of the tech doc, which will soon be due.

2.13 Week 5, Winter

With the looming deadline of midterm checkin this week involved a lot of reassessments of our documentation. We each planned out what we would need to revise in our respective sections and prepared for our additional progress documentation. At the same time we continued to put work into our research, with Zach focusing on perfecting the drone communication, Dominic making more malware packages, and I continued honing my research on sensor spoofing, waiting to test it on the fully functional drone. With midterms on top of it, next week is bound to be just a flurry of writing and editing, probably with little research getting done.

2.14 Week 6, Winter

This week, as expected, was spent almost exclusively updating our documentation and making the changes necessary. I've finished creating the OneNote document and have also made the new progress report docs. We've all added changes and edited our video. We reused parts of the video from last term that didn't change, and added in more current updates. Next week we can finally be done with all this writing and be back on the research cycle.

2.15 Week 7, Winter

After a solid term of difficulty after difficulty with the drone setup we decided to seek the help we had long needed. Zach spent a lot of time writing up specific documentations on what he had done and tried so that we could get the best possible recommendations. In the mean time I've been working on getting a unified and structured from of data recording up and running. I spoke with Kirsten to get some direction on it, and she gave me some great ideas. With this in place we can all put our research findings in one place, and will be able to see how the others are recording so that we can maintain some sort of consistency. This also means that when the time comes to bring all of our results together and draw conclusions, it will be a much easier process.

2.16 Week 8, Winter

2.17 Week 9, Winter

I had meant to also get all of my person research and data gathering put up, as well as the packages I've found or created, but unfortunately my mountains of other homework overtook that. I'll be putting them up as I can throughout the next week. After we've met with Kevin we'll all have a better idea of how we are doing moving forward.

2.18 Week 10, Winter

Dead week, as always, consisted mostly of hurried work and and scrambling to meet deadlines. For this class there was just preparation work for our upcoming individual reports and then the final presentation. We each worked on putting the terms worth of work together so that we could best present and discuss it. I spent a good deal of this time preparing for a job interview, and left most of my prep work to be done over the weekend, which it was. Going into next term is looking pretty solid, we just need to keep up the work ethic and be better about documenting what we find.

2.19 Week 1, Spring

As with most first weeks, this first week of Spring term has consisted of assessing and planning. We met with Vee and made some plans for what we need to do moving forward, specifically on getting the poster done and getting our research data together. I've made progress on updating our threat models and creating designs for our final poster look. We seem to still have some problems with our work being disparate but with more meeting it should be resolved soon. Once we know that we're all on the same page with the same game plan we can really get things moving.

2.20 Week 2, Spring

Week 2 was a productive one, we had a much needed meeting in which we hashed out exactly where we need to be going in the last week of this project, and how we all understand our responsibilities within it. We've made plans to work together more often and to put all of our heads together to grind out as many exploits as we still can. As for progress, I made some on my Mavlink script, and I also updated the threat model. The poster draft was also imminently due so I updated it and added graphics. Miscommunication was a continuous problem in the project and I think that's be fixed now after our meeting, but now that a lot of what I thought was my important work has been cut down I have the problem of filling that space with a lot more code or other forms of vulnerability proof.

2.21 Week 3, Spring

This past week involved Herculean work effort, pushing what I had been working on, planning for the next couple weeks, and exploring new exploit topics. Reevaluating what needed to be done before expo gave me a great chance to make new in depth plans. Specifically they involve focusing all my efforts between now and May 1st on pushing any software exploits I've been working on, and quickly developing the OS level ones I've recently started. Some of this time will also need to be spend on perfecting our final poster draft. After the code freeze takes place on the first my focus will shift more towards expo prep and data analysis.

I've made a good deal of progress so far on pushing out my code by finishing my second GPS spoofing package and conceptually finishing my MavLink package. I've started work on three separate OS level memory manipulation exploits, and hope to finish them by the end of this week. Getting these tricky OS exploits to work in a week is bound to be problematic, I'm hoping to get some help from Kevin when that inevitably happens.

2.22 Week 4, Spring

This week involved a LOT of work. I had planned to work on some OS exploits and have spent quite a lot of time on them, with some solid progress being made. Now that those and the poster are so close to being finished the plans for the next couple weeks move to our WIRED writing assignment and to preparations for expo. We also plan to continue making progress on the code base, fixing up any packages that need love and documenting them all. As expected there were a number of problems in getting our poster beautifully strapped together, which unfortunately required some last minute scrambling. Better planning for the upcoming portions of the work will hopefully prevent much more of this from happening.

2.23 Week 5, Spring

Week five was spent polishing up the exploits we had, and preparing for our whitepaper and midterm report. We made plans to get together and put as much work as we all can into our whitepaper, to get the best possible result. We've also planned to record our progress video and pitch together. As for work done this week, we each wrote our WIRED papers reviewing a member of another group, which was an interesting and useful exercise. It gave me good ideas for how to pitch our project to someone, with the mindset of them reviewing it. This was a surprisingly problem free week, mainly since it was somewhat of an eye of the storm in term of overall project development.

2.24 Week 6, Spring

This week was somewhat odd in that we were updating and tidying up our repo, but there weren't any actual assignments going on. We did get the actual deadline for our whitepaper at the end of the week so that gave us the cue to spring into action. Thankfully I had already laid out something of an outline for it and had a lot of the writing already done in different places. It wasn't too hard to piece together a bare bones draft of it over the weekend. Our biggest problem was that we were flying somewhat blind as to what the final product should look like. There isn't really an official whitepaper structure for us to follow.

2.25 Week 7, Spring

This week was the long awaited expo week! We spent nearly all of it preparing and perfecting our whitepaper. This meant planning out how it was going to work and meeting with Kirsten to tweak the layout and organization of sections. I was really impressed wth all of our combined effort being able to create a complete and polished document by combining our preexisting writing and tying it together. We did have some problems with how to approach some sections or on whether or not some aspects should be included. I also personally struggled with the wording of some sections, so I spent a lot of time bugging Kirsten with questions.

As for expo it went really well and I honestly enjoyed the experience. We had each planned out a spiel and pitch to use when talking to differing groups of people. We were able to interact with a wide variety of backgrounds and interests, and many of them really liked our project. Personally I had the hardest time communicating it with children since I've never been particularly good with them.

2.26 Week 8, Spring

We've made it! All the hardest parts are over, and only the final written wrap ups are left ahead of us. We have plans to meet with Kirsten early next week to discuss in more specific detail what our group needs to do for the final write up, since ours is a little different. Once that happens then we'll all work to get it put together and as nicely done as possible. For the moment we've all been working on the three short writing assignments that were revealed in the final class meeting. As for problems we currently don't have any, except that we're stuck waiting for our meeting this weekend rather than being able to make progress on our final report. Final Post Questions

If you were to redo the project from Fall term, what would you tell yourself? I would definitely tell myself to start working on the unknowns, like the hardware, much sooner, rather than assuming we could easy get it done. This would reduce sooooo much of the stress and headache from this project.

What's the biggest skill you've learned? I've learned a huge amount about project and time management. I definitely got better at estimating the time things would take and my ability to get them done.

What skills do you see yourself using in the future? I see myself using the two skills I just mentioned extensively throughout my personal and professional life, and I'm sure I'll always be working to improve them.

What did you like about the project, and what did you not? While it was rough sometimes, I really liked the challenge of being able to work on a research project about something of which I had little foreknowledge. I enjoyed being able to continuously learn from it. I did not so much enjoy the seemingly excessive progress reporting, particularly in video from, but I understand why it was assigned.

What did you learn from your teammates? A lot! They both taught me a variety of different skills, and showed me different ways of thinking about things on various occasions. I think we all traded at least a little knowledge, and hopefully each learned how to operate well in a group.

If you were the client for this project, would you be satisfied with the work done? Yes, I would be. I know we've had miscommunications about what satisfactory meant previously in this project, but I know we've all put a lot of work into it, and strove to fix that and make it the best it could be.

If your project were to be continued next year, what do you think needs to be working on? Our project would be a great candidate to be continued, probably as a development project that creates fixes and patches for ROS based on our research, and help to secure the system.

3 Zach's Blog Posts

3.1 Week 0-3, Fall

Ah yes, the first blog post. We spent time trying to figure out a good time and place to meet with our client, and hasing out details. We got some input from some research professors, which helped us to refine the goals of the project. Next week we hope to get antiquated with the hardware and start planning attack vectors and threat modelling.

3.2 Week 4, Fall

We spent time clarifying some logistical things with the class, including getting feedback on our Problem Statement writeup; which we need to redo. We also got to see the drone! There's some work to be done before we can get it working, and we should be able to have it run ROS. Being that we are using ROS, we already have some vulnerabilities that we can look at, as ROS has some out of the box. We may be looking towards SROS, a secure variant of ROS, though it is still a very early project. We also had a chance to meet with Jesse, a crypto researcher who once worked for Intel. It will be awesome having his input on the project as we move further along. Next week we hope to finish up our Problem Statement and get our Requirements doc written.

3.3 Week 5, Fall

This week we finished up our revised Problem Statement document, and should be squared away on that. We also spent a lot of time trying to work out our requirements document to get our first draft submitted. We were able to meet briefly with our client to check in and make sure everyone was on the same page with regards to the requirements document. After we finish up some formatting and polishing, we should have a good starting point for moving forward.

3.4 Week 6, Fall

We had a busy week trying to put together our final version of the Requirements document. We were not able to meet with our client until the end of the week, which caused some concerns regarding our document being too ambiguous. However, being that this is a research project, with requirements that can't fully be defined until we complete our threat model. So we are hoping that we can make changes to this document later on.

3.5 Week 7, Fall

Spent this week trying to hash out tech document sections and doing a lot of research into the drone hardware and ROS. Hopefully this helps us figure out the direction our research will take us, though I feel we will run into some issues due to the open endedness of our project; it is hard to define a clear objective that is measurable.

3.6 Week 8, Fall

Spent a lot of time researching more into the drone, more specifically the communications channel. Turns out there is a lot of things that we can do with our drone, and we have no idea what additional hardware is available for us to use. It seems that our client is also unsure, though we haven't really been able to speak with him much since the requirements document was turned in. Hopefully we are able to have a meeting soon.

3.7 Week 9, Fall

Due to unanswered questions regarding our drone, I was unable to meet the tech document deadline. For example, we still have no idea how our drone will be controlled; what kind of ground control station do we have? Is it operating on the 2.4 Ghz or 900 Mhz band? How do we interface this with ROS and the Pixhawk flight controller? There are a lot of unanswered questions, and after meeting with our client I was basically told that it didn't matter and to just pick something for the sake of turning in the document. I was not, and am still not a fan of bullshitting such an important document, and find it a bit annoying that we are left to just ”wing” it, without any real direction from our client or those helping us with the project. Though, I did the best I could making education decisions, and was able to get the document further completed.

3.8 Week 10, Fall

During the holiday break, I finished up the tech document and started working with my team to get our design document figured out. It turns out we don't have to follow the IEEE template mentioned for the design document, since it is a software specific standard, and we are doing a research based project that does not involve writing some fancy app or what have you. Speaking with Kirsten allowed us to get more direction on how to properly format our document. After much hard work we finished our design document.

3.9 Week 11, Fall

During the weekend leading up to finals week, our group worked to get our progress document completed, and began work on our powerpoint presentation. We also discussed talking points for our video that we have to make. It looks like we will make the deadline Wednesday, which is good. We are basically on our own at this point, so hopefully we made the right calls in our design document, and Winter Term starts smoothly. Over winter break, myself and Emily plan on meeting in the design lab to get our drone working with ROS, and to do some testing to see if we can use the SROS (Secure ROS) project at all. Things are getting exciting!

3.10 Week 1, Winter Break

The start of winter break has been used for final exam recovery, and not much project stuff has been done. I looked into what we need to do to get our drones operational so we can start threat modeling. Lots of snow and ice closed campus preventing me from using the capstone lab. I'm hoping I can start on things next week, and that my other group members are available.

3.11 Week 2, Winter Break

I spent time making sure the BeagleBone Blacks were operational. Also picked up two SD Cards so we can easily boot up ROS and whatever else we need, without needing to flash the local memory. I have not got ROS running just yet, but at least we know that our hardware seems to be working as it should. Once I get ROS running with our drone packages installed, I will see if I can get the PixHawk cape to work properly.

3.12 Rest of Winter Break

Planned out meeting with our groupmates for the first week of winter term. I continued to get our microcontrollers ready for use, and inquired about getting some SD cards from Kevin McGrath. Also inquired to get our drone controllers, so we can actually get things operational.

3.13 Week 1, Winter

Got our hardware together and organized, as we get ready to get the drones running. Kevin McGrath gave us the pairing cables so we can link the drone controllers to each of our drones. Met with Emily and Dominic to discuss our next steps, and plans to finish our threat modeling. We still need to meet with our client, to go over what we have and to get further direction for the remainder of the term.

3.14 Week 2, Winter

McGrath gave us some microSD cards for the BeagleBone Black microcontrollers – I worked to get Ubuntu and ROS running on both boards. Also did some testing to make sure it was working correctly. Next week we plan on meeting up to see if we can get the Pixhawk interfaced and working.

3.15 Week 3, Winter

Finally got Ubuntu ARM running on the BeagleBones, along with a working install of ROS!!! It took two all nighters to work through some issues, but everything finally fell into place. We also met with our client, Vee, a couple times this week to discuss our next steps. I was also able to test the PixHawk Cape; we are able to read and write memory via I2C, which is good. Plans over the weekend include getting the dones actually hooked up and seeing if we can get them fully operational. Exciting things ahead!

3.16 Week 4, Winter

Got ArduPilot compiled on the Beagle Bone Blacks; we need to setup the PWM connections on the drone, to actually test if this works. This process took several hours, for each BBB. If this fails, we will roll back to an earlier version of Ubuntu and ROS. Week 5 will be when we figure this out, and get the drones flying, finally.

3.17 Week 5, Winter

Still having issues regarding the ArduCopter project, seeking assistance trying to get this error message resolved: http://diydrones.com/forum/topics/can-t-connect-to-mavlink-via-usb-panic-ap-baro-read-unsuccessful Speaking with Vee our client next week about this.

3.18 Week 6, Winter

After talking with our client, and working on our midterm progress, it was suggested that we go to Kevin McGrath regarding next steps, since we are not getting anywhere getting our drone configuration working. We have solid work done towards ROS packages and exploitation regarding our project; we just don't have hardware for it to run on. The next step during week 7 is to figure out what to do regarding these issues

3.19 Week 7, Winter

This week was spent trying to figure out next steps regarding our drone hardware issues. I have been tasked with sending a detailed email to Kevin McGrath, seeking guidance. We also got initial feedback from our client Vee, who wanted to see more detail and info regarding ROS packages in our midterm progress report. We spoke with our TA, Jon, to see how to best accomplish this. Vee, Kevin, and Kirsten were all brought in the loop on this, as there was a disconnect between what the assignment asked for, and what our client was looking for. Right now our focus is getting our hardware situation worked out, and we will fine tune the documentation later on. We hope to meet with Kevin McGrath next week to help figure out what to do with our hardware. I am also working on communication exploit related ROS packages, after speaking with Vee briefly via Slack on Friday night (24 Feb 2017) – This is the first time I have been told to work on ROS packages, even though Dominic has been working steadily on them for the past several weeks. I hope that my work towards this will help with getting our project moved forward, while trying to work out our real-world hardware issues. There was also some talk about going towards new hardware, or a virtual robotic environment. Either way, we are still focusing on ROS exploitation, and have been making solid work towards that goal.

3.20 Week 8, Winter

We had a morning meeting with McGrath to talk about our issues getting the BBB and Pixhawk 1.6 cape working with our drone. McGrath had a lot of ideas and was incredibly helpful! He provided me with access to a new BBB image to try, that already has the BBB + PixHawk cape device trees setup for you. This Erle Brain image is no longer published online, so we were very thankful that Kevin McGrath had a copy of it to share with us on Box. We also spoke with Kevin about our policy exemption request, so that we can sniff 2.4 Ghz and 900 Mhz RF traffic while on campus – I was given details on what to include in the document to formally request that exemption. We also had a good discussion regarding methods to intercept this RF traffic, and talked about various SDR (Software Defined Radio) options that we could use – some cheap and some not so cheap. After the meeting with McGrath I went to the capstone lab to flash the new BBB image to test it – And I am very happy to say that it appears to be working!! The rest of this week will be spent trying to hook up the drone again and see where it leaves us.

3.21 Week 9, Winter

This week I wrote up the policy exemption document for McGrath – The OSU Security Council is meeting Monday of Week 10, and discussion of this request is formally on the meeting agenda fingers crossed that it all goes well! We all met in the capstone lab a few times this week to hook up the drone and see where we are now that we have a working BBB image. We can communicate with our Mission Control software, but only for a moment before we lose contact. We are going to request a pair of MAVLink radios and try this again. Also during our tinkering, we found that our GPS cable for the 3DR radio doesn't have anywhere to go on our PixHawk 1.6 cape – it appears the cable we were given was for a newer version of the Pixhawk, and no the one we have. After Dominic and Emily spoke with Kevin about this, he gave us another cable to try – he too was a bit perplexed by this. We will be meeting up

early next week to give things another go to see where we are. We also had our final capstone lecture for the term, where we did simulated pitches of our project. We were chosen to do ours in the front of the class, and got some solid feed back doing so. We need to have some solid drone visuals to keep people interested an engaged. At this time we are not sure exactly how to do that, as it seems that we can't have the drones at the expo. Next week we will be finalizing our end of term documents, including our poster – the deadline for that was dropped on all of us last minute, after being told it would be due at the beginning of next term. But hey, what's a little more stress and pulling your hair out, right? Oh, I also came across this awesome research paper by the U.S Air Force, regarding exploiting the MAVLink protocol
VULNERABILITY ANALYSIS OF THE MAVLINK PROTOCOL FOR COMMAND AND CONTROL OF UNMANNED AIRCRAFT
DEPARTMENT OF THE AIR FORCE
http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA598977

3.22 Week 10, Winter

I was really sick at the beginning of dead week – plagued with a stomach flu and multiple days of trying to manage a 102 degree fever. Because of this my productivity was nonexistent at the beginning of the week. Luckily around Wednesday evening I started to feel better, and was able to start working on end of term documents. Our group worked on our poster, and finished up the draft design of it Thursday night. We also made progress on our final progress reports for the term, and discussed logistics regarding our video we need to do next week. So far progress is smoothly being made as we approach the end of Winter term. Over spring break I plan to do more work on the drone, and getting into more exploitation analysis.

3.23 Week 1, Spring

We are using this "3P" format now, so here it goes: Progress: Logistics and planning for trying to figure out where we are with the project, and what we still need to do; since Expo is next month, we are running out of time, so this stuff needs to be figured out. We spoke with Vee, our client, for about 15 minutes; luckily we have plans to meet next week, on a day where our schedules aren't so tight with classes. Plans: Come up with a game plan from now until Expo – Contact Vee, our client, and setup a time to meet to talk more in-depth about the project. Problems: Apparently my request to get a policy exemption was scrapped and someone brought up the idea of using ROS Bag; I was not informed of this until speaking with Vee during our 15 minute meeting. This really changes things as most of my communications threat model is up in the air, unless if we can use ros-bag in a way that will gather the data I am looking for.

3.24 Week 2, Spring

Progress: We were not able to meet with our client until Friday, and that's when shit kinda hit the fan. The client decided earlier to throw out the drone, though I was not informed of this when that decision was made. With that, we set a game plan on how to proceed.
Plans: Work on ROS only Proof of Concepts, that do not need a drone to show that vulnerabilities exist. Meet with client next Thursday to explain progress of things.
Problems: Lots of uncertainties, and the project seems to be on a fine line at the moment.

3.25 Week 3, Spring

Progress: Spoke with several people this week about the state of the project, and how we should proceed. We have a great support network and have been able to find some direction in this last minute chaos. Vee, our client, was pleased with my progress on my ROS PoC packages, so things seem to be going in a positive direction now.
Plans: Continue work on PoC packages, document everything, and get poster feedback so we can finish it.
Problems: Still some uncertainties regarding how the project is going.

3.26 Week 4, Spring

Progress: Made my ROS fuzzer this week, and started work on my ROS replay attack package. Submitted poster for final approval before going to the printers on Monday.
Plans: Finish up remainder of ROS PoCs, for Monday's code freeze. Finalize poster and get client approval.
Problems: I was sick with strep throat this week, so I was not able to get as much done as I wanted.

3.27 Week 5, Spring

Progress: We got our poster done this week, and managed to get a pretty cool team photo for our poster (and proudly displayed on the Github repo).
Plans: Start working towards Midterm progress report and video
Problems: No problems to report currently; our client still seems to be happy with the new direction of work.

3.28 Week 6, Spring

Progress: Worked on and finished the Midterm progress report and video. Finished first draft of our white paper. Hopefully we will get feedback before the Expo due date.
Plans: Finalize whitepaper next week, and prepare for Friday's Engineering Expo
Problems: Got last minute notice that our whitepaper is expected to be submitted by Friday, which is expo day. Why we got such late notice on this, I do not know. We will do what we can to work within this limited time frame.

3.29 Week 7, Spring

Progress: We produced our final white paper in time for expo. Expo went well, and it was a great experience!
Plans: Start working towards final report, and other final writing documents.
Problems: No problems to report.

3.30 Week 8, Spring

Progress: Got information regarding our final assignments for the term to wrap up our capstone experience. Started working towards completing some of those assignments.
Plans: Scheduled a meeting with Kirsten to talk about our Final Report requirements, since we have a research project.
Problems: No problems to report.
Final Post Questions If you were to redo the project from Fall term, what would you tell yourself? I would tell myself to identify exactly what needs to be done much sooner; there was so much time spent trying to navigate what to do first. Getting those things out of the way sooner would have made things less stressful.
What's the biggest skill you've learned? Dealing with conflict. Unfortunately this project hit a lot of problems along the way, and things often got tense between myself and other team mates, as well as tense with our client. Knowing how to navigate situations like this is important, as it can often get in the way of success.
What skills do you see yourself using in the future? Project planning, team work, and conflict resolution.
What did you like about the project, and what did you not? I really enjoyed evaluating ROS security in a lab setting, and developing proof-of-concepts. I disliked being tasked with getting our dated drone hardware to work, however. That caused a lot of issues.
What did you learn from your teammates? Learned how to better communicate my progress with certain tasks, and saw different ways of approaching problems.
If you were the client for this project, would you be satisfied with the work done? Yes, I would. Assuming that I was better at communicating and checking in with my group along the way; there were some communication issues with our client and to be honest I have no idea if he is pleased with the work that was done after we worked through these issues.
If your project were to be continued next year, what do you think needs to be working on? A continued evaluation of ROS, as well as a revamp of the current PoCs we have; a lot of the packages could use some work. Also, getting these PoCs tested on real world hardware, like a drone. It would also be good to spend some time working on fixes

for the security issues that we discovered.

COLLEGE OF ENGINEERING

Electrical Engineering & Computer Science

SECURITY FOR ROBOTICS

Researching Vulnerabilities in the Robotic Operating System

Why is this Important?

THE PROBLEM WITH ROS

The Robotic Operating System (ROS) has a lot of problems from a security standpoint due to being middleware. Middleware does not typically focus on security because it needs to be lightweight.

Without any specific security measures taken, there is room for all manner of attacks. To try to tackle them, the vulnerabilities were broken into three categories:

1. Those affecting the confidentiality of data
2. Those affecting the integrity of data
3. Those affecting the availability of data

CREATION OF A THREAT MODEL

A threat model was created to visualize the three categories. It serves as a roadmap for the research and can be looked back on at any time. The model below in Figure 1 is organized as a tree, with the three main branches being the three components of the CIA triad (Confidentiality, Integrity, and Availability). This way one can look at our threat model, decide if their concern is one of confidentiality, integrity, or availability, then examine the vulnerabilities in that category.

Not every area from the threat model was successful, some were not possible to breach, but it's still good to acknowledge that they are still likely targets.

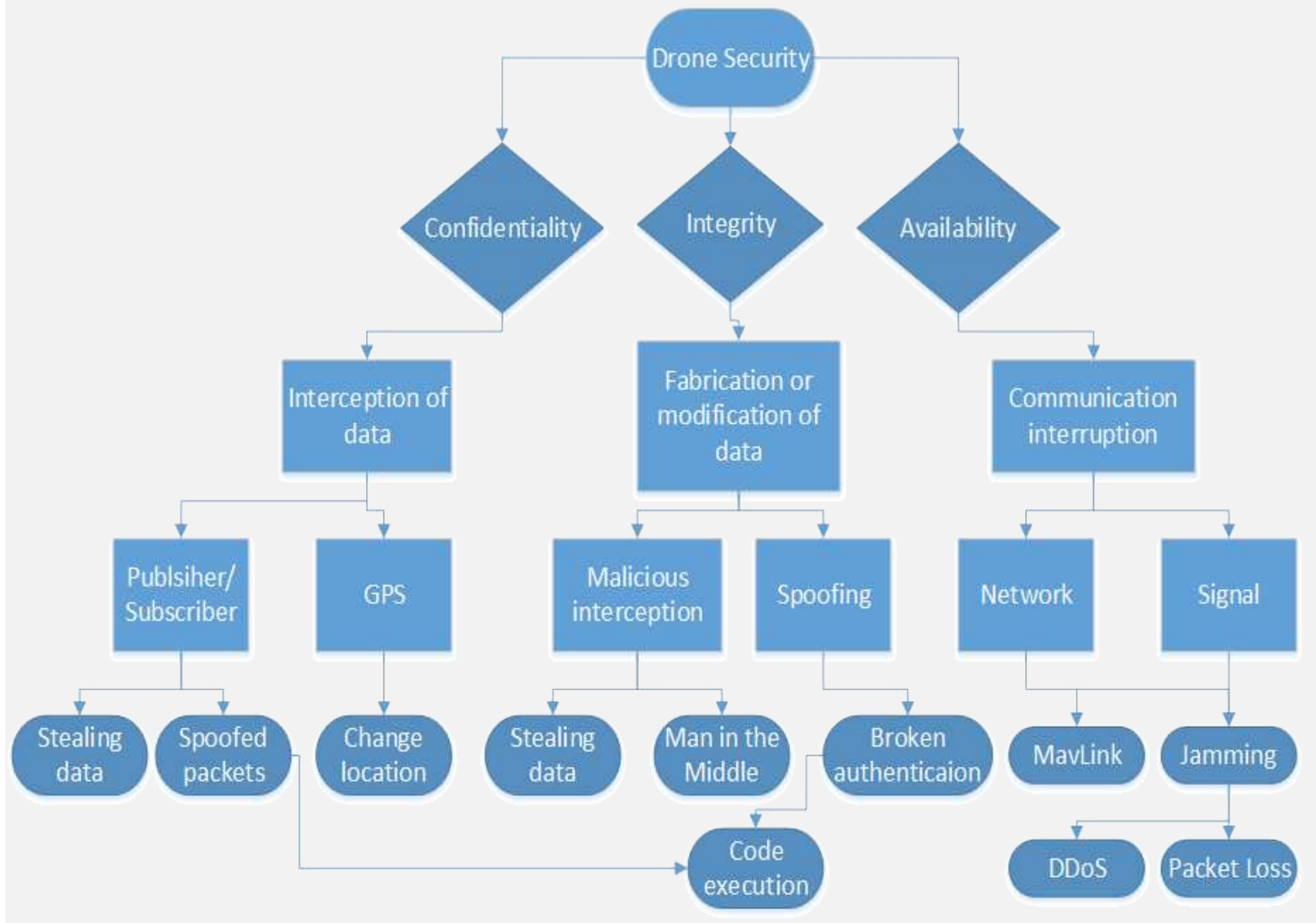


Figure 1: The three pathways of ROS vulnerabilities and how they can be exploited

RESEARCH DESIGN

The project aims to find exploits for whichever vulnerabilities seem most the most open. This means first doing some research to locate the best starting points. Because each exploit is unique it is difficult to draw final conclusions as well as define which exploits are the most severe.

Failure Mode Effects Analysis (FMEA) is a procedure used in a variety of fields which creates an empirical system for testing and logging any failures, which can be applied to all of our attacks. Somewhat akin to risk analysis, FMEA involves defining severity, occurrence, and detection rating scales, usually from 1 to 10. After these scales have been defined one can use them to calculate a risk priority number (RPN) and a criticality number with which the found failures can be ranked in order of importance. You can see this represented in figure 2 below.

As for the sort of exploits created in this project, there was a variety of paths covered. Some examples of these attacks are:

- Man in the Middle
- Spoofing
- Broken Authentication
- Denial of Service
- Resource Consumption
- Data Stealing

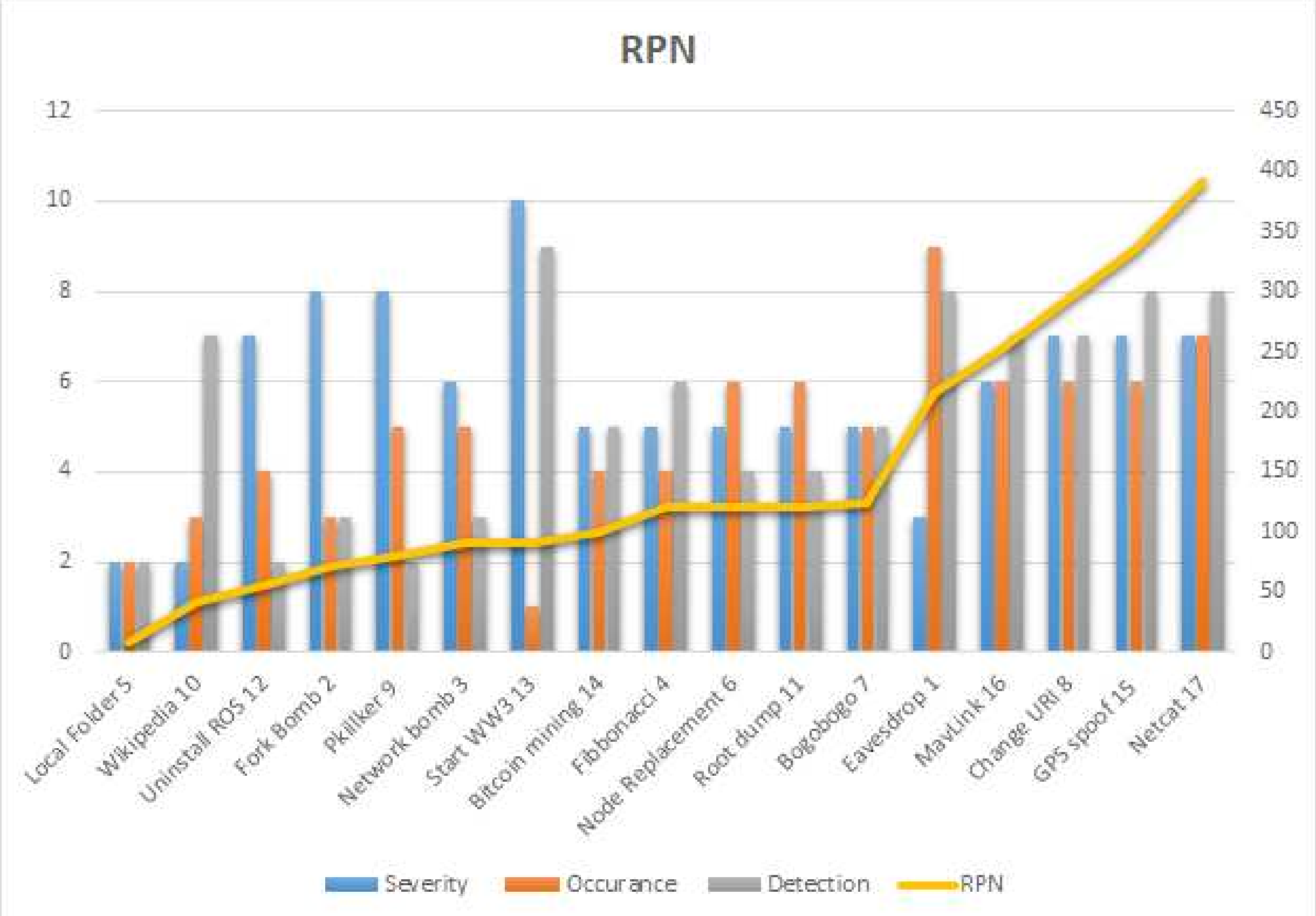


Figure 2: This chart shows that the RPN is a more meaningful number than the others alone

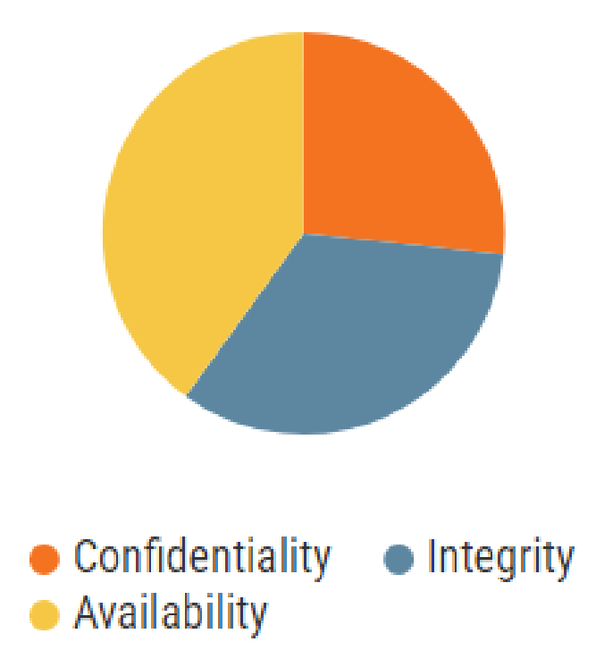
NOTABLE PROOF OF CONCEPTS

- Lack of Authentication
  - Remote control over ROS
  - Package Installation
- Process Communication (Publisher/Subscriber Model)
  - Fuzzing of Subscribers
  - Data Capture of Publishers
    - ROS Bag Replay Attack
  - Sensor Spoofing
- Denial of Service
  - Bogo Sorting
  - Bitcoin Mining

The Exploits

30

Total



The CIA Breakdown

The 30 exploits cover a wide range of vulnerability types and use various methods of attack. Eight of them compromised confidentiality, ten of them imperiled integrity, and twelve of them affected availability.

Figure 3: View at a glance of which areas of security our work affects

OUR FINDINGS

Just as suspected, ROS has wide range of vulnerabilities that this project exploits with a success rate of nearly 100%. Most of these exploits involved either entering the system via the publisher/subscriber system that ROS uses to communicate, or demonstrated what malicious actions could take place on the system once inside. In figure 3 above is a visualization summing up the numbers in our research.

We also found that our project evolved as we went along. We had initially intended to do more hardware based attacks, but they proved to be too cumbersome and difficult to generalize. There are also a board range of physical attacks ranging anywhere from an EMP type gun to an eagle taking down an airborne drone.

The Team:

WHO WE ARE

We are a group of students interested in cybersecurity, guided by our client's graduate research project.

Team Members:

Emily Longman

longmane@oregonstate.edu

Dominic Giacoppe

giacopped@oregonstate.edu

Zach Rogers

rogersza@oregonstate.edu

WHERE IT GOES FROM HERE

We hope that this research will be continued and expanded upon not only by our client in his thesis, but also by other members of the security and robotics communities. The more that is done in this field, the better all future robotics can be, without being a huge security threat.

Client:

Vedanth Narayanan

narayave@oregonstate.edu

CS Capstone Research Paper

June 13, 2017

Security for Robotics

Prepared for

Oregon State University

Vedanth Narayanan

Prepared by

Group 50
RoboSec

Emily Longman
Zach Rogers
Dominic Giacoppe

Abstract

In drones and other networked robotics there is a broad array of security vulnerabilities that can be leveraged in an attack. We will evaluate ROS to find as many of these security holes as we can and document them. The different vulnerabilities found will be categorized into malware, sensor hacks, network and control channel attacks, and physical breaches. For some of these exploits we may be able to implement solutions, which will also be documented. These findings and any solutions will be added to an ongoing academic effort to make robotics more secure.

Contents

1 Introduction

Robotics is still a relatively up and coming field and as most efforts in robotics are in pursuit of furthering capabilities, security has been left largely untouched. Because of this many functioning, deployed robots have limited to no security in their internal systems, making them very vulnerable to attacks. While the community developing robotics is still largely academic and there is little worry of being attacked, the security vulnerabilities still exist. Soon robotics will become ubiquitous in society and hackers will exploit these vulnerabilities for personal gain. There have already been reports of smart appliances being used for botnets, it's only a matter of time before drones and other robotics are similarly abused. [1] This project aims to eliminate some of this abuse before it begins.

2 Problem Statement

Since robotics are continually in development and there is such a wide variety of devices, focus needed to be on a small subset of robotics to have any hope of making progress within a year. Because of this the Robotic Operating System (ROS) was chosen for this project. These operating systems can be attacked at the driver level with malware, they can have the configuration files modified, and they can have data intercepted or spoofed in their internal communication subsystems. We could also investigate the ability to spoof sensor data, such as the camera, IR guidance, accelerometer, or gyroscopic systems. Outside of the OS level there is a huge amount of room for exploitation in the communication and control channels used by networked devices. It's very important that we acknowledge all of these risks, since exploitation of them could be a big setback for global trust of robotics.

Essentially the problem is twofold; first a specific feature of a specific device needs to be isolated, and then that feature needs to be broken, and all work documented. This could be a crucial improvement for the use of robotics running ROS in a wide variety of sectors. The military uses and plans to use them widely, and for them more than anyone they need to be as secure as possible. Amazon and other commercial shipping companies have a vast use of robotics, and we've all heard about their plans for delivery drones, which are a huge security risk. If consumers think these will be abused they won't trust this upcoming technology and it will be slowly or never adopted, which will make companies hesitant to invest in their development. Consequentially, the world as a whole will be slower to advance robotic technology. Hopefully our work in securing robotics can help to better develop consumer trust in this emerging technology.

3 Literature Review

This project would have been much more difficult without a variety of documentation and previous research into the security of ROS and robotics in general. One paper that was widely used in the creation of our project basis was "A Preliminary Cyber-Physical Security Assessment of the Robotic Operating System (ROS)". [2] This article was written by a group of researchers who created a honeypot robotic system and brought it to DefCon 20 to let hackers there exploit it as they could. This lead the researches to be able to recored a range of attack vectors from a wider variety of minds. They also used a number of their own methods to study what was being done, such as examining the packets being sent and received through wireshark, visualizing the effects with hardware, and using Backtrack Linux. Their work provided starting points for many different ways to attack the system, and for cataloging the effectiveness of each different type, relative to the others. The inspiration from this paper laid a groundwork for the threat models that would be created here, and for the exploratory structure that was followed.

Another particularly useful paper was written by Bernhard Dieber et al, which discussed security on the application level of ROS. It provides great insights for the problems with authentication and data integrity that are present in ROS. [3] This article helped to provide guidance for that portion of our model, and was foundational in the understanding of how the application level interacts with the rest of the system, from a security standpoint. A similar paper entitled "ROS: an open-source Robotic Operating System" was vastly useful in the planning of this project, as it gave a detailed overview of the system from a research standpoint. While the official site and documentation of ROS was absolutely invaluable, this paper provided us with a third party assessment of the system and how all portions work together. Without a thorough understanding of this research would be much more difficult. [4]

The last of the papers that were used extensively in the overall planning and implementation of this project was one discussing the SROS project. This project is very closely aligned with ours in that they are working to bring better security to ROS in its most vulnerable sectors. The areas in which this project most wanted to provide security were some of the first that our project investigated. Specifically it provided insight into the access control policies needed in ROS, the insecurity of the communication channels, and they way in which processes are handled. [5]

When looking for background information for specific exploits that would be created, a lot of other research papers, journal articles, and security presentations were examined. These are referenced and discussed within each of packages to which they contributed.

4 Threat Model

A key element to our project is being able to accurately find vulnerabilities or identify areas where they are likely to be. One of the best ways to do this in an organized way that can be later referenced is with threat modeling. Adam Shostack has a well defined method for doing this, which was followed to develop the model for this project. [6]

An important thing to note is that ROS is middleware, which means it sits between the base Linux OS, and the application level.Typically there is little focus on security because middleware like this needs to be lightweight. Without any specific security measures taken, there is room for all manner of attacks. To try to tackle them, the vulnerabilities were broken into the three categories of the CIA Triad [7]:

- Those affecting the confidentiality of data
- Those affecting the integrity of data
- Those affecting the availability of data

A threat model was created to visualize these three categories. It serves as a roadmap for the research and can be looked back on at any time. The model below in Figure 1 is organized as a tree, with the three main branches being the three components of the CIA triad (Confidentiality, Integrity, and Availability). This way one can look at our threat model, decide if their concern is one of confidentiality, integrity, or availability, then examine the vulnerabilities in that category.

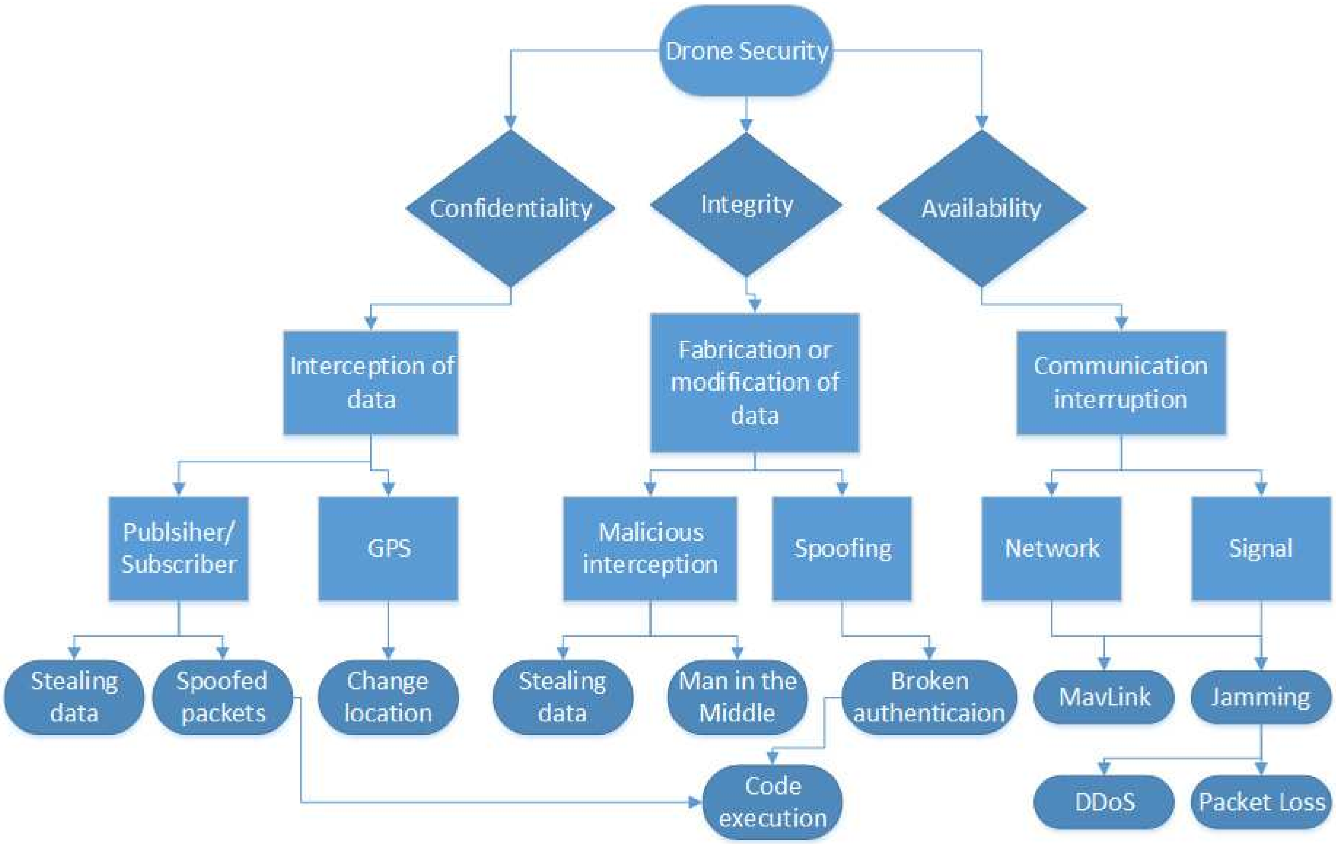


Figure 1: A view of the attacks that are viable against the ROS system

In this figure the top levels are the three sectors of the CIA Triad, which branch into the attack types within each. Below that level are the rectangular examples of vulnerabilities of that type. Lastly the bottom tier nodes are attack type examples.

5 Packages and Solutions

As mentioned above, the exploits written were meant to cover a lot of ground topic wise. Some are based on the same vulnerability, while others are more distinct. Specific packages have been chosen below to represent each of the notable categories of attack. These range from resource consumption, to network sniffing and spoofing, and to OS level exploitation. In the Analysis section below there is a table showing where in the CIA triad each of the exploit packages falls.

5.1 Eavesdropper

Called Malpac in the repo, this is a very simple package. It has 2 nodes, one that publishes a string every couple seconds, and a subscriber that takes that message and prints it to stdout. There is also a third malicious node which also subscribes to the publisher and prints the same message with a slight modification to stdout as well. While not that malicious in this case, the general principle can be used to receive and change any published data, including sending said data to other computers using the ROS subscriber/publisher system and compromise the original datas integrity. For example, if you had a drone doing aerial reconnaissance of an area and transmitting the images back to a base, this kind of system would be able to intercept those images.

This package was one of the first ideas implemented so it started in one of the most obvious places within our model, the Publisher/Subscriber system. It falls under the confidentiality section of the threat model because it steals data that would be best kept confidential. Although not particularly sophisticated, the unencrypted nature of this communication channels means it doesn't need to be. With little difficulty it demonstrates how trivial it is for a malicious actor to listen in on communications between nodes and steal useful information. This is expanded upon in later packages which take this easily gained information and use it spoof the system.

5.2 Forkbomb

This package is a forkbomb. As stated previously there is no process monitoring or control native to ROS, so this is a very simple forkbomb. All it does is do a little math to slow things down a bit, then call itself 8 times and exit. Eventually there are enough processes to clog the CPU and cause a kernel panic, bringing down the robot. In theory, as long as there is no existing form of process management, this would kill any robot with enough time, which has obvious use cases.

Forkbombs like this one work to fill up processing resources so that the system has a much harder time scheduling legitimate processes. This is essentially denial of service, which falls into the availability branch of attacks, since the system resources are rendered unavailable when done successfully. Distributed denial of service (DDoS) attacks are becoming more and more common in Internet of Things devices, and this package shows how it easy is to do something similar within a ROS device.

5.3 Network Bomb

This exploit temporarily disables the operating system's wireless networking capabilities. For a robot dependent on wireless communication for control or navigation, this has some rather obvious consequences. Depending on how the robot handles the lack of networking it could do anything from fall out of the sky to automatically returning home. It could also permanently disable the wireless networking with a small tweak, which would make the only fix to physically capture and wire back into the robot to fix the networking. Either way, if one's robot is dependent on wireless networking, it will be downed for a bit.

Much like the forkbomb, and similarly named, the network bomb is also an exploit that reduces the availability of the system resources. Specifically this one uses up the communication resources, which cuts it off from the control center and potentially allows an attacker to take over. This was developed to flesh out the DoS portion of the model, so that the forkbomb focuses on consuming internal resources, while this one consumes the network.

5.4 Fibonacci

This one is actually our Client, Vee's package. It does a naive recursive implementation of the calculation of Fibonacci numbers, and tries to calculate the 100000th number. It either takes a long time, or stack overflows and crashes, neither of which being terribly great results and both using a fair amount of system resources. As such it has the potential to bring down the system, depending on how exactly python handles the recursion involved, or at least be a waste of resources for a bit.

This was one of the first packages in existence and served as somewhat of an example of what future packages might look like. It was also why a number of the earliest packages focused on resource consumption and computationally intensive work. Even if this isn't the most creative or greedy way to reduce availability, it serves as an elegant and classic proof of concept for availability focused packages.

5.5 Local Folder

This package just creates and then deletes a local folder. In theory it can delete any user level folder, which may include program files or ROS nodes or maybe just something important locally. If it has super user permissions then it could go so far as delete system directories or the like, but that is not guaranteed. Either way it can probably delete something that the user would rather it not.

This package is yet another one that compromises the availability of the system. While it could be used to remove mundane things, it also would be capable or deleting crucial folders, giving it the potential to be quite dangerous. It is an interesting approach to the availability side of attacks, since they often use tactics that increase the load on the system, rather than removing critical portions.

5.6 Node Replacement

Consisting of 2 parts, this includes one package which launches a publisher that publishes a string every couple seconds and a subscriber that takes that string and prints it to stdout. The other one runs a package that kills the subscriber from the 1st package, and replaces it with its own subscriber with the same name that prints a modified message to stdout. At the time we thought the node would be indistinguishable from the previous node besides the different message, but it turns out that ROS silently assigns each publisher and subscriber its own unique ID and is in fact quite noticeable. It might be hard to notice by an inattentive user, assuming they do not notice the slight hiccup where the old subscriber dies and is replaced, so it still has potential to be a threat or phone home.

This exploit works on the integrity portion of the tree, by doing a rudimentary form of a man in the middle attack. The system is somewhat fooled into thinking that the new node was the original publisher and takes information from it. While the unique ID makes this package not useful as is, it still provides a solid proof of concept for the idea. It was also used as a starting point for some later iterations of the same concept.

5.7 Costly Sorting

This is a package that performs bogo-bogo sort. [8] Bogo-bogo sort is an extremely inefficient sorting algorithm that performs a random shuffle to sort a list, recursively. While it is not particularly system intensive, the sort for a list with a mere 10 elements can take on the order of days. The goal of this package was not to be directly malicious but more of a lurking threat; it shows that ROS also does not check for any particularly long running processes, which allows for a variety of actual threats. It also wastes some CPU time. The best example is an attack that lies dormant until signaled to do maximum damage, or one that waits for a specific process to kill it.

This package falls into the availability spectrum, as it uses a processing intensive sorting algorithm to eat up computing resources. As stated above it also serves as proof of concept for the fact that ROS doesn't cut processes off after any certain amount of time. This same idea can be used in many different ways, with different types of resource consumption being called at specific times to deny service to legitimate actions.

5.8 URI Change

This is a package that changes the ROSMASTER URI to an arbitrary one, and starts a new ROS master to match. The URI is basically where packages check for the ROS master node, which is the master in the ROS master/slave system. ROS master manages nodes, and the publisher/subscriber system between nodes as its 2 main duties. As such, changing the URI and starting a new ROS master node means that all new nodes will talk to the new ROS master as opposed to the old, which could allow for some interesting node management or fiddling with the publisher/subscriber system. The ROSMASTER URI can also be set to a remote machine if it setup correctly as well, which means you could have a compromised robot respond to a remote ROS master node as well, allowing an attacker to control the robot remotely.

5.9 Pkiller

This simply pkills (process kill, for those unfamiliar) all processes with ROS in the name. This will at least kill ROS master, and maybe some other related ROS processes, but killing ROS master also kills all the nodes currently being managed by ROS master, so it will also kill all ROS processes on the system. This has the obvious effect of bringing the robot to a halt, and has the potential of doing major damage. Just image if a drone carrying an explosive payload fell out of the sky because its control system suddenly stopped.

This attack removes availability by killing key processes, but can also be used to kill something legitimate and then replaces it with a malicious process, which would be an integrity attack. Much like many of the previous packages, this one was created to further explore the ways in which system control and accessibility can be taken away, with something other than just resource consumption.

5.10 Massive Download

This package uses wget to attempt to download all of Wikipedia. It probably will not succeed, unless the robot has enough disk space to hold all of Wikipedia, but it will fill the disk up to the limit. This can cause a number of issues depending on what else needs disk space, but it is sure to cause some sort of issue. If nothing else it will prevent any more ROS processes from being launched, as ROS master does some record keeping for each process and will need a little disk space for that purpose.

This package works to break down the availability of the system by filling up disk space, which mean it can stop legitimate data being stored. This is an interesting approach to denial of service in that it doesn't have to be continuous but rather only needs to be run once. After having done this there are a number of routes the attacker could follow, such as replacing this downloaded data with their own malicious files, or even making the initial massive download contain malware to serve whatever purpose they desire.

5.11 Root Dump

This package dumps your filesystem to a remote machine, currently a dummy address. If you had something confidential on the robot, it is now in the hands of some other party, assuming your network connection did not die mid-transfer or something. If your drone had those top secret recon images, or maybe the drone's code itself was supposed to be a secret, they could potentially be in the hands of some unknown party.

This package is one of the first in the confidentiality portion of the tree. It uses surprisingly simple tactics to give a malicious person access to a copy of the entire file system, leaving them free to peruse the contents and use them. This goes to prove just how big even basic file encryption on ROS would be, since that would prevent this from being a viable attack.

5.12 Uninstaller

This one, assuming your ROSMASTER was run with root privileges and is on a Linux distribution that uses apt-get as its package manager, uninstalls ROS from the robot. The robot's operations will be unaffected until next boot as the ROS binaries will be loaded into system memory, but on reboot or if ROS is killed and restarted in some other way, ROS will no longer be there. Especially bad if the ROS install was customized, as one would have to redo all those configurations all over again...

This attack is almost the exact opposite of the previous one. It uses system capabilities to remove ROS itself, which is essentially the ultimate shut down. Although, as stated above, this wouldn't actually take effect until the next time the system was booted up, it still serves as a very effective way to deny someone access to their robotic control by rendering their system completely unavailable.

5.13 Start WW3

This pings (what the team knows to be, but can not directly confirm) a North Korean IP address. While it has no direct effect on the system, alerting North Korea to your presence is not exactly a great idea in the current political environment, and has a good chance of getting one on a watchlist.

This package was written mostly for fun, since it doesn't do much from a research standpoint. It's hard to even classify where it would fit into the CIA Triad. It seems to hurt the person who owns the ROS device more than the device itself. In that case it would be quite effective for targeting someone and getting them investigated or arrested or in some way causing them legal and diplomatic problems.

5.14 Bitcoin Miner

This package attempts to install a bitcoin mining service on the robot. It does currently need superuser permissions for this, as it does need to use apt-get, which means that ROSCORE needed to be run as a superuser so that this child process would have those permissions as well. If the install succeeds however, then you have a bitcoin miner using CPU time for someone elses monetary gain, and with a little more work could be further configured to have the miner run at boot as a system process, but for the purposes of the package it was good enough as is.

Bitcoin mining is also a more fun type of package, but it does also serve to reduce the availability of the system resources since its mathematically intensive. It also somewhat touches upon the other two portions of the triad in that it compromises the system's integrity, and reveals some potentially confidential system information.

5.15 ROSless Listener

Here was an attempt to abuse how ROS subscribers and publishers really worked outside of ROS. All subscribers and publishers actually communicate through a shared TCP/UDP socket controlled by ROS master, and the general idea was to make a non-ROS socket to communicate with the ROS socket, subscribe to a known publisher on the ROS machine, and print whatever they published as plain text to stdout. In short, something like the first malpac, but going around ROS to do so. Unfortunately ROS uses TCP/UDP with a special header encoding that makes this difficult, as to communicate any publisher or subscriber you need their unique ID, which is assigned to them by ROS master at their launch. And to get said ID, you need to do some handshaking with ROS master, which turned out to be a bit beyond our technological capacity and time constraints. We believe it is still possible to achieve this, but to do so would require a better working knowledge of sockets and more investigation into the handshake protocol used by ROS master. While the documentation is available online, specifically on the technical overview page of the official ROS site [9], it was just a little too much for the team with the time we had. Malpac ROS is similar to this, but differs in that Malpac ROS still uses ROS to achieve this goal, while this package aimed to bypass ROS entirely.

5.16 Linux Kernel Exploitation

This package contains two scripts, a memory hook and a stack smasher. The first, the memory hook, is actually a Linux kernel exploit, and doesn't use ROS itself, but ROS commonly sits atop Ubuntu or Ubuntu-based linux systems. [10] This package does a simple memory hook into the syscall table, which allows the script to make changes to whichever syscalls it wants once it finds the syscall table memory address. [11] This is done by changing the pointer to the function of an existing syscall and replacing it with a pointer to a malicious function. This is the kind of script that would likely be secondary to one of the network attacks. One of those might be used to get into the system, and could then install LKMs like this one. Since it operates at the kernel level in Linux, ROS would have no way of ever knowing something was amiss, which renders this a very insidious threat.

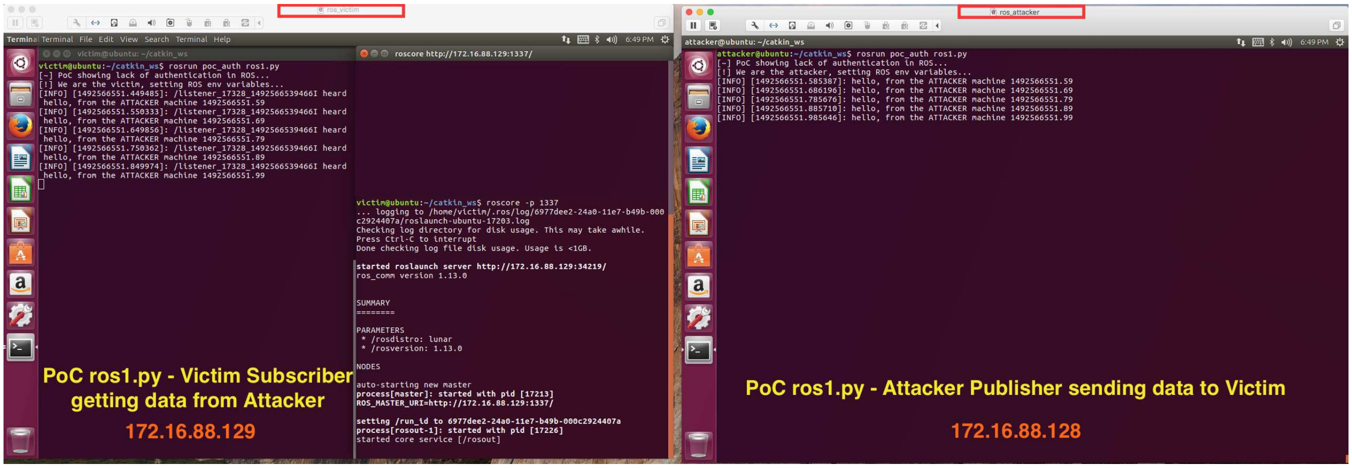
The second script, the stack smasher, provides a basic proof of concept (PoC) for stack smashing. [12] This is a form of buffer overflow attack which has been used reliably for a very long time to gain unauthorized access to a system. It works by leveraging the overflow to push its way into privileged space and then execute some form of malicious code once it has entered that area of the stack. [13] This specific script is a small scale buffer so better demonstrate the idea. To use a simple approach like this, one needs access to the environment variables of the system so it's not perfect.

Both of these different OS exploits affect the availability and the integrity of the system. They both remove the availability of their area's primary functionality on the way to modifying it and damaging the integrity. Some confidentiality is also lost in the process, but it's a negligible effect. The idea for both of these came primarily from subject matter covered in the Defense Against the Dark Arts class, and because the underlying kernel was an area of the system that had yet to be touched.

5.17 GPS Exploitation

The exploits in the category work with the GPS or Mavlink protocol. Specifically the GPS package uses both spoofing and man in the middle to trick a drone into thinking it is the home base, and can misguide it. This is done by sniffing for drone network information, and then leveraging that to send spoofed landing data via UDP. The Mavlink script follows the same idea, but it was never able to reach a point of development beyond simply sniffing for the information and positioning itself as a node in the middle. With better documentation and a system with which to reliably test it the Mavlink script could grow to the point of the GPS one.

5.18 ROS Broken Authentication PoC


ros_victim
Terminal Terminal File Edit View Search Terminal Help
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
6:49 PM
ros_attacker
attacker@ubuntu: ~/catkin_ws
PoC ros1.py - Victim Subscriber getting data from Attacker
172.16.88.129
PoC ros1.py - Attacker Publisher sending data to Victim
172.16.88.128

```
victim@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the victim, setting ROS env variables...
[INFO] [1492566551.449485]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.59
[INFO] [1492566551.550333]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.69
[INFO] [1492566551.649856]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.79
[INFO] [1492566551.750362]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.89
[INFO] [1492566551.849974]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.99
```

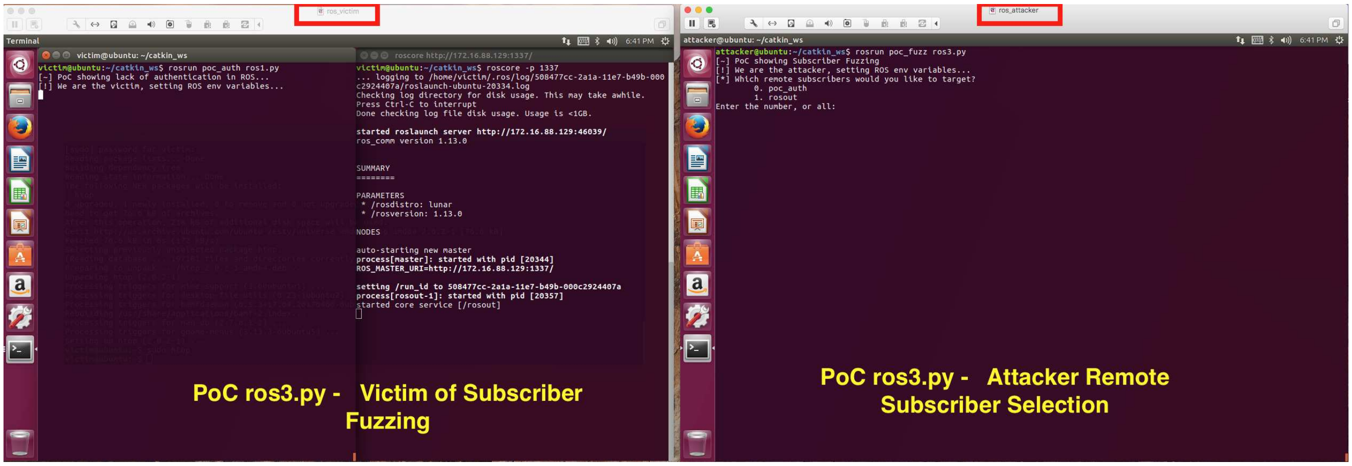
```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/6977dee2-24a0-11e7-b49b-000
c2924407a/roslaunch-ubuntu-17203.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:34219/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [17213]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 6977dee2-24a0-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [17226]
started core service [/rosout]
```

```
attacker@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the attacker, setting ROS env variables...
[INFO] [1492566551.585387]: hello, from the ATTACKER machine 1492566551.59
[INFO] [1492566551.686196]: hello, from the ATTACKER machine 1492566551.69
[INFO] [1492566551.785676]: hello, from the ATTACKER machine 1492566551.79
[INFO] [1492566551.885710]: hello, from the ATTACKER machine 1492566551.89
[INFO] [1492566551.985646]: hello, from the ATTACKER machine 1492566551.99
```

Figure 2: PoC ROS Broken Authentication

Two PoC packages were created that show ROS has zero authentication measures, by leveraging the publisher subscriber model. These simple yet effective PoCs are the basis for remote ROS exploitation. If you know the IP address of a remote ROS machine, you can leverage it as you wish, without the need to authenticate with the machine in any way. This should not be taken lightly, as this opens up ROS to any device that has a network connection. The PoC proving this to be true connects to a remote ”victim” machine, sending data to a ROS subscriber process. This complete PoC package extends upon Malpac 8 and 15. The idea came from expanding upon a study done by a university in South Korea [14].

5.19 ROS Process Communication


ros_victim
Terminal
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
6:41 PM
ros_attacker
attacker@ubuntu: ~/catkin_ws
PoC ros3.py - Victim of Subscriber Fuzzing
PoC ros3.py - Attacker Remote Subscriber Selection

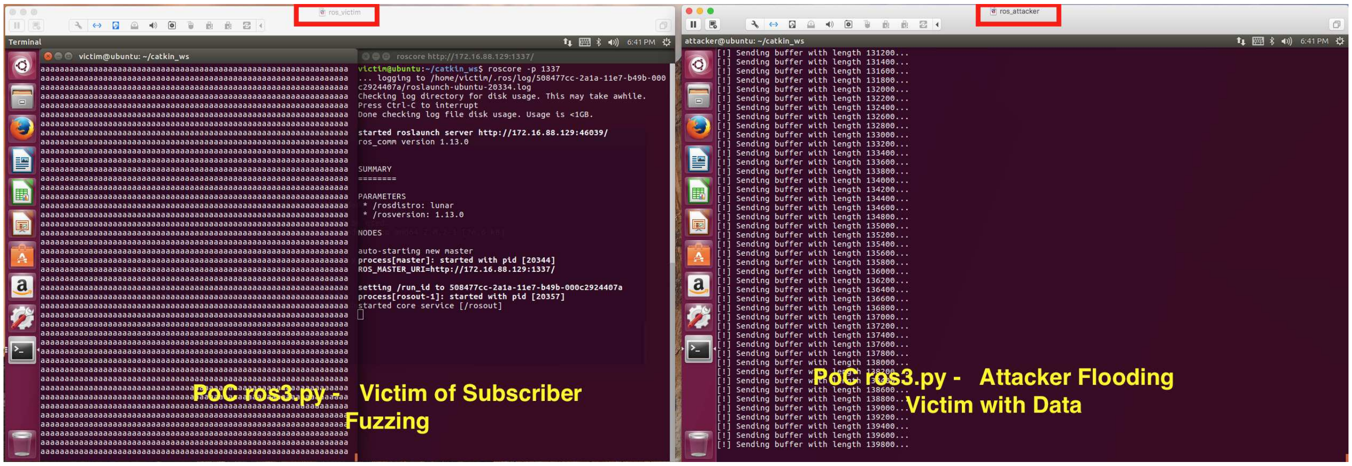
```
victim@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the victim, setting ROS env variables...
```

```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/508477cc-2a1a-11e7-b49b-000
c2924407a/roslaunch-ubuntu-20334.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:46039/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [20344]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 508477cc-2a1a-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [20357]
started core service [/rosout]
```

```
attacker@ubuntu:~/catkin_ws$ rosrun poc_fuzz ros3.py
[~] PoC showing Subscriber Fuzzing
[!] We are the attacker, setting ROS env variables...
[*] Which remote subscribers would you like to target?
        0. poc_auth
        1. rosout
Enter the number, or all:
```

Figure 3: PoC ROS Fuzzer Selection

Two additional packages were created, exploring exploitation of the process communication model that ROS uses. These PoC packages act as ROS security tools, one of which is a subscriber fuzzing tool, and the other a publisher data capture tool. The subscriber fuzzer allows an attacker to target remote ROS processes by flooding them with large amounts of malformed data. This can be used to expose issues with ROS processes on a real world device, like a drone. The idea of creating a fuzzer came from a study that looked at using a similar technique to exploit the MAV Link communicaitons protocol [15]. The remote publisher data capture tool uses a ROS tool called rosbag to create a saved instance of a given ROS process. This PoC uses that data to preform what's known as a ”ROS Bag Replay Attack”, which can cause devices running ROS to carry out a given operation at the will of the attacker. For example, an attacker could capture what happens with a drone flight control process turns on the motors to begin flight. The attacker could then ”replay” that action remotely, causing the drone to fly. It is also possible to modify this captured data before replaying it via ROS, so an attacker could replay a malicious payload quite easily using this method. The idea of the ROS replay attack came from a South Korean study [14].


ros_victim
Terminal
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
6:41 PM
ros_attacker
attacker@ubuntu: ~/catkin_ws
6:41 PM

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/508477cc-2a1a-11e7-b49b-000
c2924407a/roslaunch-ubuntu-20334.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:46039/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [20344]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 508477cc-2a1a-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [20357]
started core service [/rosout]
```

PoC ros3.py - Victim of Subscriber Fuzzing

```
[!] Sending buffer with length 131200...
[!] Sending buffer with length 131400...
[!] Sending buffer with length 131600...
[!] Sending buffer with length 131800...
[!] Sending buffer with length 132000...
[!] Sending buffer with length 132200...
[!] Sending buffer with length 132400...
[!] Sending buffer with length 132600...
[!] Sending buffer with length 132800...
[!] Sending buffer with length 133000...
[!] Sending buffer with length 133200...
[!] Sending buffer with length 133400...
[!] Sending buffer with length 133600...
[!] Sending buffer with length 133800...
[!] Sending buffer with length 134000...
[!] Sending buffer with length 134200...
[!] Sending buffer with length 134400...
[!] Sending buffer with length 134600...
[!] Sending buffer with length 134800...
[!] Sending buffer with length 135000...
[!] Sending buffer with length 135200...
[!] Sending buffer with length 135400...
[!] Sending buffer with length 135600...
[!] Sending buffer with length 135800...
[!] Sending buffer with length 136000...
[!] Sending buffer with length 136200...
[!] Sending buffer with length 136400...
[!] Sending buffer with length 136600...
[!] Sending buffer with length 136800...
[!] Sending buffer with length 137000...
[!] Sending buffer with length 137200...
[!] Sending buffer with length 137400...
[!] Sending buffer with length 137600...
[!] Sending buffer with length 137800...
[!] Sending buffer with length 138000...
[!] Sending buffer with length 138200...
[!] Sending buffer with length 138400...
[!] Sending buffer with length 138600...
[!] Sending buffer with length 138800...
[!] Sending buffer with length 139000...
[!] Sending buffer with length 139200...
[!] Sending buffer with length 139400...
[!] Sending buffer with length 139600...
[!] Sending buffer with length 139800...
```

PoC ros3.py - Attacker Flooding Victim with Data

Figure 4: PoC ROS Fuzzer Execution

6 Analysis System

Failure Mode Effects Analysis (FMEA) is a procedure used in a variety of fields which creates an empirical system for testing and logging any failures, which can be applied to all of our attacks. [16] Somewhat akin to risk analysis, FMEA involves defining severity, occurrence, and detection rating scales, usually from 1 to 10. After these scales have been defined one can use them to calculate a risk priority number (RPN) and a criticality number with which the found failures can be ranked in order of importance. You can see this represented in figure 5 below.

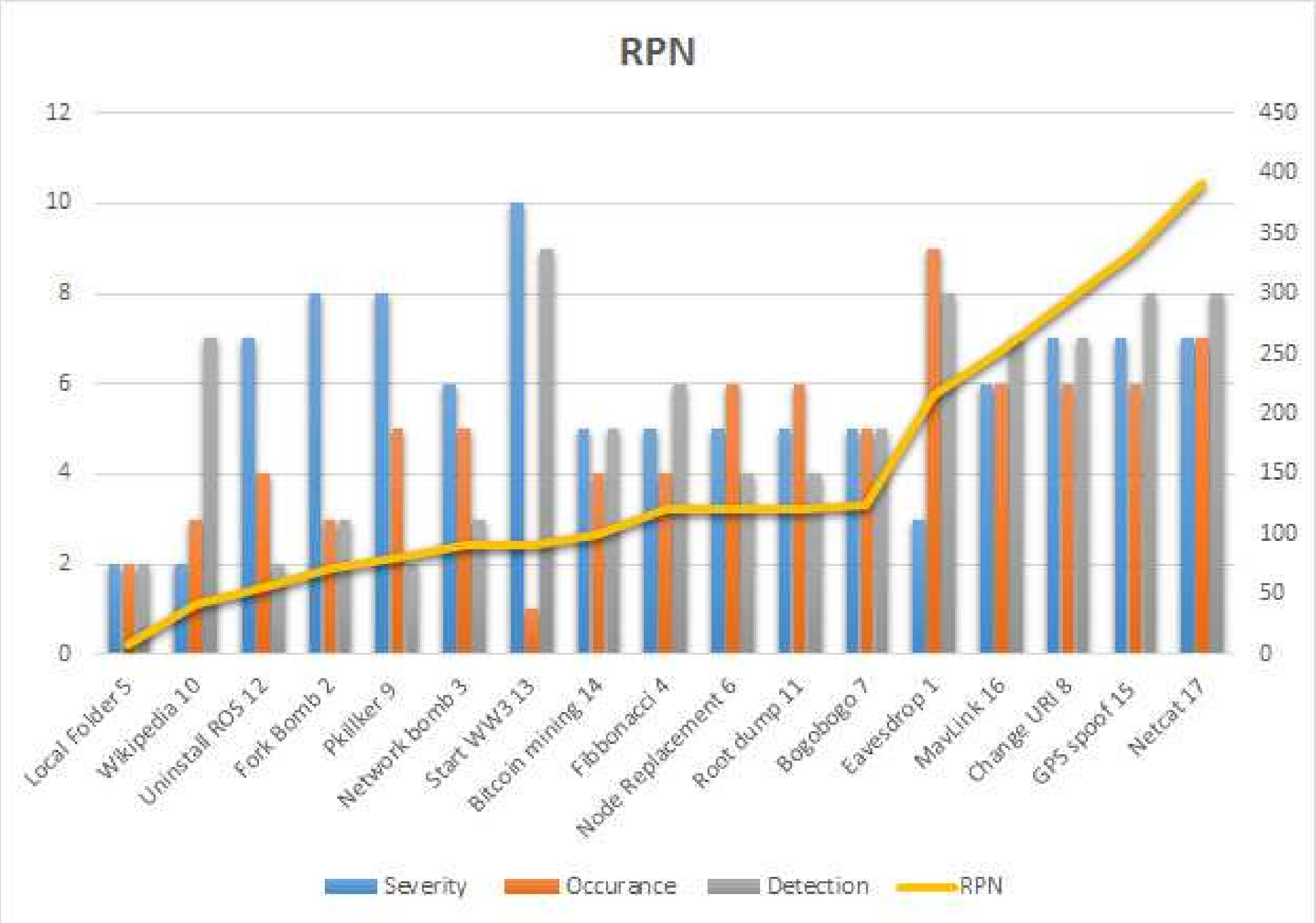


Figure 5: The increasing criticality ratings of packages, based on the three combined numbers

7 Findings

Just as suspected, ROS has wide range of vulnerabilities that this project exploits with a success rate of nearly 100%. Most of these exploits involved either entering the system via the publisher/subscriber system that ROS uses to communicate, or demonstrated what malicious actions could take place on the system once inside. Those that interrupted the availability of the system were most prevalent, since it's somewhat trivial to break subsystems once inside. The next most common type of exploit was those that tamper with the integrity of data, usually in network communication. The least common were those that broke the confidentiality of data, often through authentication or sniffing.

We also found that our project evolved as we went along. We had initially intended to do more hardware based attacks, but they proved to be too cumbersome and difficult to generalize. There are also a board range of physical attacks ranging anywhere from an EMP type gun to an eagle taking down an airborne drone.

Not every area from the threat model was successful, some were not possible to breach, but it is important to acknowledge their importance. This new version of the threat model was created to showcase the success level of each area using color coding. Those areas that are red have a lot more potential for vulnerabilities, yellow has been covered by our research but is still likely to have more, and green was heavily covered and only an expert in that area would be likely to identify more. This also includes our specific packages as leaves, rather than the general concepts they covered.

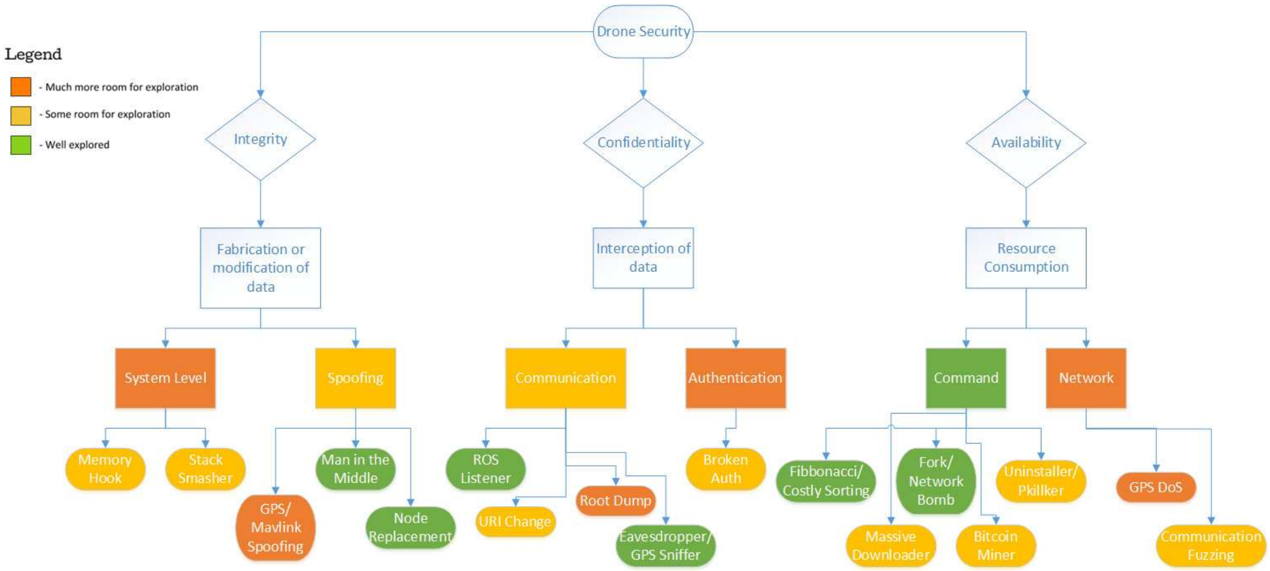


Figure 6: map of the areas which were most or least covered in this research

8 Analysis

As discussed above in the Methods sections, FMEA was used primarily in the analysis of this project, in order to discover which of the created packages would be most catastrophic. In the figure above representing the graph of the RPNs, one can see which types of attacks were ranked the highest. Those that focused on the integrity of the system turned out to have the highest concentration in the top of that ranking.

There are a number of factors that could have lead to this clustering. Firstly, the majority of them were more thorough and written in a greater depth than many of those in other categories. This means that they would be likely ranked more severely since they do more damage in more detail within the scope of this research. Another potential reason for this is that spoofing and changing of data to something malicious tends to be perceived as more nefarious than something like resource consumption.

Another interesting metric by which to compare the exploits is the time it took to write them. On average the time to complete an availability package was 18.75 minutes. For confidentiality the average was 246.7 minutes. Integrity came in at 296 minutes average development time, which is the longest. This helps to back up the trend of integrity focused packages being ranked as some of the most severe. If more time went into the development of them, then it makes sense that they would likely be more sophisticated.

The table from which these averages came can be viewed below.

Package Name	Primary Type	Time Spent (in Minutes)
Malpac2	Availability	10
Malpac3	Availability	20
Malpac5	Availability	5
Malpac7	Availability	25
Malpac9	Availability	20
Malpac10	Availability	20
Malpac12	Availability	15
Malpac14	Availability	35
Malpac1	Confidentiality	30
Malpac11	Confidentiality	25
GPS1	Confidentiality	300
Malpac_Coms (nc_drop.py)	Confidentiality	720
Malpac_ROS (ros1.py)	Confidentiality	360
Malpac_ROS (ros2.py)	Confidentiality	45
Malpac6	Integrity	40
Malpac8	Integrity	30
Malpac13	Integrity	10
Malpac15	Integrity	300
GPS2	Integrity	420
MavLink	Integrity	240
MemHook	Integrity	840
StackSmash	Integrity	540
Malpac_ROS (ros3.py)	Integrity	480
Malpac_ROS (ros4.py)	Integrity	60

9 Limitations

The main limitation our project faced was not being able to do testing on actual robotic hardware. If this project were continued, the next steps would be to setup a ROS enabled drone and see how our exploit packages effect the drone's functionality. Testing on real world hardware will drive home the real-world implications of using an insecure middleware on robotic devices with open communication channels. Our research is not just limited to drones; any device running ROS can be used to further test our methods. While our testing was done on ROS enabled machines, those machines were not real-world robotic devices, thus the real-world impact of our packages could not be evaluated.

If this project had spanned longer than one year there also would have been much more time to expand the breadth and depth of this research. Keeping our investigations within the scope of both the project requirements and the time constraints was a continuous challenge.

10 Future Research

There were plans to use real world hardware to test our ROS exploit packages, in this case a drone. However, we ran into some issues getting our hardware platform to work. This section will try to go through the hardware that was used, as well as the issues that occurred along the way so that the same mistakes are not made in the future. Also, where possible, there will be a mention of recommended next steps for anyone interested in continuing this project.

There were two drones at our disposal, though we really focused our efforts on one of them; the Flame Wheel ARF F550. The other drone we had was a complete, proprietary NazaM2 based drone, with its own micro controller and flight controller. The Flame Wheel F550 was already hooked up to a Beagle Bone Black, which is the micro controller we decided to use, so it made sense to focus on the Flame Wheel drone. Along with this Flame Wheel drone, we had a Beagle Bone Black micro controller, a Pixhawk v1.6 Firecape, and a 3DR compass and magnetometer radio. The idea was that ROS would be running on the Beagle Bone Black, along with the ArduPilot project [17], and the Pixhawk would act as our flight controller. Based on the current ArduPilot project documentation, the Pixhawk is a supported flight controller [18]. However, we later found out that our Pixhawk v1.6 was no longer supported, and documentation was incredibly lacking. Our team recommends that the newer version of the Pixhawk be used instead of the out-dated v1.6.

It should also be mentioned that we were given two Beagle Bone Black micro controllers, and two Pixhawk v1.6 flight controllers. Only one of the Pixhawks was used, as the other one had damage to the tty0 connection. We had planned to fix this ourselves, though never got around to it. Also, only one of the Beagle Bone Blacks was used, as there was no reason to suggest that it had any problems.

10.1 Why This Hardware Was Used

We decided to use this hardware as it was available to us and did not require that we purchase anything new. In terms of why use a drone, we discussed the current trend of drones becoming popular in the hobbyist and commercial sectors, and thought it be important that our security evaluation of ROS extend to drones that use it. Moving forward, we feel there is no reason to use different drone hardware, as it is our view that the issues we ran into are related to the micro controller and flight controller. We would highly suggest that newer versions of the supported micro controllers be used, along with a compatible paired flight controller, that is still supported by the manufacture and the community. These include, but are not limited to, the Erle Brain 2, the newer Pixhawks, and more [18].

Due to the issues we encountered, we were not able to test our exploits on real hardware. Instead, we managed to lay the ground work needed for future work to be extended to real world hardware, such as drones. It is our view, that great care and time needs to be taken to find a set of hardware that is currently supported, and that works with the ROS drone configuration we were hoping to achieve.

10.2 Beagle Bone Black: Stability, UbuntuARM, and ROS

The first thing we did with regards to getting our hardware setup was to get a stable operating system running on the Beagle Bone Black (BBB). We knew that we wanted to run ROS, and according to the ROS documentation, Ubuntu was the route to go [19]. To do this, we followed the extensive documentation to install Ubuntu ARM 16.4, which was the latest Ubuntu ARM build at the time [20].

While flashing Ubuntu on the BBB was pretty easy, we ran into an issue where the BBB would randomly restart and would enter user flashing mode, even though we flashed Ubuntu to the internal memory. After going through this process a second time, we managed to get a stable Ubuntu ARM version running on the BBB. The logs for the internal eMMC flash can be seen on our project GitHub [21]. We are not sure why this did not work the first time, but after a couple weeks made the decision to reflash Ubuntu, which ended up working as expected.

Once Ubuntu was running, we installed ROS following the online documentation [22]. This process went as expected, and did not present any issues. Once we had ROS running on a stable Ubuntu build, we moved on to try and get the BBB to detect the Pixhawk v1.6 Firecape.

10.3 Pixhawk i2c Detection and Beagle Bone Device Trees

Following along with the ArduPilot documentation, after installing Ubuntu and hooking up the Pixhawk flight controller, running the 'i2cdetect' command should tell you if the Pixhawk is being detected by the BBB [23]. When we ran this command, and we did not get the same output [24], which told us that the Pixhawk was not yet being read by the Beagle Bone Black. UbuntuARM did not seem to have the device trees preconfigured for the Pixhawk.

Looking ahead, we found that the ArduPilot project has the device trees needed for the Pixhawk Fire Cape [25]. We thought that by moving forward with compiling ArduPilot, we would also be working towards getting the Pixhawk operational.

10.4 Compiling ArduPilot Project

After installing the needed build tools on UbuntuARM, we proceeded to to pull the latest version of the ArduPilot project from git, per instructions listed in the documentation [26].

After the code was pulled from the ArduPilot git, we compiled the ArduCopter specific packages, as this is needed for multi-copter, or hex-copter style drones, which is what our Flame Wheel F550 is. A log from this process can be seen on our project's GitHub [27].

It took a long time to compile the ArduCopter code on the Beagle Bone Black, and all appeared to compile correctly, without any errors.

We then continued with the instructions on building and enabling the proper Beagle Bone Black device trees for the Pixhawk Firecape, and enabled the cape manager at the kernel level, per these instructions [25].

It should be noted, that these build instructions are assuming that the Beagle Bone Black is running a Debian image, not an Ubuntu image. We continued using the Ubuntu image at this point, as we saw it as needed in order to get ROS working properly, per the ROS build guide [19]. It is unclear if this was the reason for our initial issues with getting the BBB to communicate with the Pixhawk.

10.5 ArduPilot Pixhawk Detection Issues

At this point, even after using the device trees from the ArduPilot project, the Pixhawk still was not being recognized by the Beagle Bone Black. Feeling stuck, we spoke with one of our project mentors, Kevin McGrath, who suggested we use an ”all in one” ArduPilot image, that has everything we need ready to go. Upon our initial research, we could not find such an image, though luckily our mentor was able to share the Beagle Bone Black image with us.

It was an old version of the ErleBrain image that supported the Pixhawk Fire v1.6 cape; the current image on their website is for the Pixhawk Fire Mini and Raspberry Pi. Flashing this image instead of UbuntuARM showed results immediately. Upon first boot the Pixhawk started to respond to the Beagle Bone Black, and ROS along with the ArduCopter project was ready to go. We highly suggest that this image be used at as a starting point in the future, as our group spent well over a month trying to sort through issues with getting the UbuntuARM image and device trees to work properly. We have a copy of this image on our project GitHub [28].

10.6 Pixhawk Wiring and 3DR GPS Incompatibilities

According to the ArduCopter documentation, we need to have a Pixhawk flight controller, RC Receiver, 3DR GPS Unit, an assembled multi-copter drone, and a battery [29]. Since we had all of these, and it appeared we now had a working image flashed on the Beagle Bone Black, we started the process of hooking everything up, per the directions outlines in the ArduCopter documentation. This is when we started to run into more problems, as the documentation shows a wiring guide for a newer version of the Pixhawk flight controller, and not the 1.6 version that we had [30].

One of the primary issues we faced was that there was no port on the Pixhawk that was labeled for the GPS unit, and our wiring for the 3DR GPS unit did not seem to be compatible with our version of the Pixhawk; after hooking everything up, there were no open ports on the Pixhawk for the 3DR GPS unit. After speaking with Kevin McGrath about this issue, he too was puzzled by the 3DR GPS not having a proper set of wires for our Pixhawk flight controller. He gave us a new wire to use for the 3DR, and we hooked it up to the only port it could plug into on the Pixhawk, the tty0 port.

10.7 Pre-flight Saftey Check Failures

For the sake of testing, we launched the ArduCopter application and it started without any problems, and signaled that it was ready to communicate with our Mission Planner software. Our mission planner was able to communicate with the drone, and was able to pull settings from the Pixhawk Fire cape. However it would then crash due to not being able to complete the pre-flight safety checks. The particular error we got with regards to this failure was that it was unable to detect our GPS unit. That error message can be seen here, where others using similar hardware had the same problem. We were not able to fix this, and think it might be due to the Pixhawk being out of date, though are not positive [31] [32].

It was at this point that our client decided that too much time was being spent trying to get the drone to work, and that we should focus our efforts on developing ROS packages to explore our threat model. It is our hope that this information will help to guide any future work on this project, so that the same mistakes are not made trying to get the Pixhawk configuration to work. We highly recommend that efforts be focused on using recent flight controllers that are still supported by the ArduPilot project.

11 Conclusion

ROS is vulnerable in at least the ways detailed above, and as such is insecure. There are also vulnerabilities whose existence we are confident of, but were unable to write code to exploit in the given time. In particular, due to our inability to repair our test drone in time, we were unable to attempt any exploits involving hardware. Most of the vulnerabilities come from basic design choices and philosophy in ROS itself, and to attempt to fix said vulnerabilities would take fundamental changes and re-designs in ROS. There are indeed already undertakings to do just that, specifically SROS and ROS2, but overall it is probably better to design drone security around the fact that ROS is insecure, if one chooses to use ROS, than attempt to fix ROS itself. Many of our vulnerability exploits come from having access to ROS itself or by abusing lax communications security to get into ROS, so just ensuring that the ROS is only accessible by trusted users is enough to prevent most exploits that we have found, although that in itself is a rather big task.

References

[1] S. Sharma, S. Garg, A. Karodiya, and H. Gupta, "Distributed denial of service attack."

[2] J. McClean, C. Stull, C. Farrar, and D. Mascareas, "A preliminary cyber-physical security assessment of the robot operating system (ros)," pp. 874 110–874 110–8, 2013. [Online]. Available: http://dx.doi.org/10.1117/12.2016189

[3] B. D. S. K. S. Rass and P. Schartner, "Application-level security for ros-based applications."

[4] M. Q. et al, "Ros: an open-source robot operating system."

[5] B. D. S. K. S. Rass and P. Schartner, "Application-level security for ros-based applications."

[6] A. Shostack, *Threat Modeling, Designing for Security.* Indianapolis, Indiana: John Wiley and Sons, Inc, 2014.

[7] I. Institute. Cia triad. [Online]. Available: http://resources.infosecinstitute.com/cia-triad/

[8] D. Morgan-Mar. Bogobogosort. [Online]. Available: http://www.dangermouse.net/esoteric/bogobogosort.html

[9] O. S. R. Foundation. Ros/ technical overview. [Online]. Available: http://wiki.ros.org/ROS/Technical%20Overview

[10] K. McAllister. Writing kernel exploits. [Online]. Available: https://tc.gtisc.gatech.edu/bss/2014/r/kernel-exploits.pdf

[11] T. Nichols. Hooking the linux system call table. [Online]. Available: https://tnichols.org/2015/10/19/Hooking-the-Linux-System-Call-Table/

[12] A. One. Smashing the stack for fun and profit. [Online]. Available: http://insecure.org/stf/smashstack.html

[13] K. McAllister. Stack smashing modern linux systems. [Online]. Available: https://www.soldierx.com/tutorials/Stack-Smashing-Modern-Linux-System

[14] K.-G. K. Goo-Hong Jung, "A study on ros vulnerabilities and countermeasure."

[15] K. Domin, "Security analysis of the drone communication protocol: Fuzzing the mavlink protocol."

[16] ASQ. Failure mode effects analysis. [Online]. Available: http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html

[17] A. Project. Ardupilot home. [Online]. Available: http://ardupilot.org/

[18] ——. Picking a flight controller. [Online]. Available: http://ardupilot.org/copter/docs/common-choosing-a-flight-controller.html#common-choosing-a-flight-controller

[19] R. O. System. Running ros on beagle bone. [Online]. Available: http://wiki.ros.org/BeagleBone#Ubuntu

[20] eLinux. Ubuntu on beagle board. [Online]. Available: http://elinux.org/BeagleBoardUbuntu#eMMC:_BeagleBone_Black

[21] S. F. Robotics. Beagle bone black emmc flash log. [Online]. Available: https://github.com/ZachR0/Security-For-Robotics/blob/master/beagle/logs/BBB_arm_eMMC_log.txt

[22] ROS. Installing ros on ubuntu arm. [Online]. Available: http://wiki.ros.org/action/show/kinetic/Installation/Ubuntu?action=show&redirect=kinetic%2FInstallation%2FUbuntuARM

[23] A. Project. Beagle bone black detecting pixhawk. [Online]. Available: http://ardupilot.org/dev/docs/building-for-beaglebone-black-on-linux.html#I2c%20Debug

[24] S. F. Robotics. Beagle bone black i2c dectection log. [Online]. Available: https://github.com/ZachR0/Security-For-Robotics/blob/master/beagle/logs/i2cdetect_BBB_drone1.txt

[25] LambDrive. Pixhawk ardupilot build guide. [Online]. Available: https://www.lambdrive.com/depot/Robotics/Controller/PixhawkFamily/PXF/Building-ArduPilot.html#Build%20devicetree)

[26] A. Project. Bbb installing and making ardupilot. [Online]. Available: http://ardupilot.org/dev/docs/building-for-beaglebone-black-on-linux.html#installing-and-making-ardupilot-on-bbb

[27] S. F. Robotics. Ardu pilot compile log. [Online]. Available: https://github.com/ZachR0/Security-For-Robotics/blob/master/beagle/logs/massive_compile_log.txt

[28] ——. Erle brain v1.1 image. [Online]. Available: https://github.com/ZachR0/Security-For-Robotics/blob/master/beagle/images/Erle-Brainv1.1-11-02-2016.img.gz

[29] A. Project. Arducopter getting started. [Online]. Available: http://ardupilot.org/copter/docs/introduction.html

[30] ——. Pixhawk wiring guide. [Online]. Available: http://ardupilot.org/copter/docs/common-Pixhawk-wiring-and-quick-start.html

[31] E. Robotics. 3dr barometer error. [Online]. Available: http://forum.erlerobotics.com/t/barometer-issue-on-erle-brain-2/1151

[32] D. Drones. 3dr barometer panic error. [Online]. Available: http://diydrones.com/forum/topics/can-t-connect-to-mavlink-via-usb-panic-ap-baro-read-unsuccessful

Learning Resources

The defacto outside resource for this project was the ROS wiki[1](wiki.ros.org), as it contains all the developer written documentation for ROS itself. It also contains some tutorials and descriptions of common tasks in ROS that the team referred to while developing packages.

For some Linux and operating system related questions, Stack Overflow[2](stackoverflow.com) continues to be a reliable source for getting things working in short order, and for some of the operating system packages it was referred to in order to answer various questions.

There were also a lot of pre-existing research that helped to guide our project, and pointed us in the right direction regarding how to tackle writing exploit packages for ROS. Most of these can be found in our Github repo [3](https://github.com/ZachR0/Security-For-Robotics/tree/master/docs/supporting_research) or discussed in the research paper.

The ArduPilot website and documentation was also helpful with regards to getting the Beagle Bone Black and Pixhawk v1.6 Cape flight controller interfaced and talking, though also showed that we really should have been using the newer flight controllers and not the outdated Pixhawk v1.6 [4](http://ardupilot.org/dev/docs/building-for-beaglebone-black-on-linux.html). This documentation also helped us figure out how to wire all the components on our drone and get the Mission Planner software configured [5](http://ardupilot.org/copter/index.html).

When it comes to books, we only really used one hard copy book, which was for the development of our Threat Models. This book was called textitThreat Modeling, Designing for Security by Adam Shostack. [6] All other reference sources used were journals or websites, and these are discussed more extensively in the research paper.

Lastly, our team had an abundance of on campus resources. Kevin McGrath was our go-to concerning attempting to get the drone operational, at least until the team gave up on that. Dave Nevin was also a great resource helping to point us in the right direction while making sure we operated within OSU's Security Policy. Kirsten Winters was crucial in our document and research design, along with Jonathan Dodge for some general steering and TeX related issues. And of course, our client Vedanth Narayanan was a great resource, along with generally steering the project to his liking.

References

[1] R. Developers, wiki.ros.org.

[2] S. O. Users, stackoverflow.com.

[3] https://github.com/ZachR0/Security-For-Robotics/tree/master/docs/supporting_research.

[4] A. P. Developers, http://ardupilot.org/dev/docs/building-for-beaglebone-black-on-linux.html.

[5] http://ardupilot.org/copter/index.html.

[6] A. Shostack, *Threat Modeling, Designing for Security.* Indianapolis, Indiana: John Wiley and Sons, Inc, 2014.

1 What Did We Learn?

1.1 Dominic

As far as technical information is concerned, I (re)learned many things about ROS and Linux, in particular creating and building ROS packages along with ROS process control and the publisher/subscriber system. However, I feel that the real learning for me was in non-technical areas, particularly communication and scheduling. First and foremost this project reinforced the idea that if Im producing work for someone, they should see it before the deadline, so they can tell me what I missed ahead of time instead of after. This was less of an issue for me personally then for the team as a whole, especially with our documentation, but the point stands. Our whitepaper was not up to snuff, nor our Spring midterm documentation, nor our design document and tech review, not even the blog posts apparently, and all of that could have been avoided if someone had looked it over before submission and told us we were completely off base. However, none of this happened because we tended to cut real close to the deadline on all of these documents, hence scheduling: starting in earnest ASAP instead of half-heartedly starting it and only really getting to it with 2 days to spare would have made it much easier to present a full draft to those in charge before the deadline instead of just getting it submitted with hours to spare. Much of the delay for those papers had to do with factors outside of my control, which was unfortunate, but the point remains that better planning could have helped with this. The biggest thing I learned about project management is that if you arent managing the project, its best to assume nobody is until evidence is provided otherwise. Most of spring term felt like I was attempting to get everyone involved on the same page trying insure the team knew what was expected of us and occasionally moonlighting as a ghostwriter for Emily. On that note, I learned a lot about project management, or at least the interpersonal communication part, not so much the planning part as by spring term it was too late to plan. Ironically, this played out like it normally does for group work for me: Im in charge of confirming requirements and relaying information between my group and those on high, and as long as I get my share of the work done things normally play out alright from there. If I got a redo, Id first and foremost redirect this project away from research, as I still cant justify this as a research paper. I would probably just make sure Vee nipped the threat model thing in the bud before it became the purpose of the project, as neither of us expected it to work out this way. Next, I would have never bothered with the drone, as I doubt it would have been useful even if we had gotten it to work and it took way too much of Zachs time. Last, I would have tried to look for a heartbleed-esqe bug in the ROS subscriber-publisher system for arbitrary code execution, because I think thats the real holy grail as far as OS based attacks. Everything I did relied on already having the ability to execute arbitrary code on the ROS system, which is a rather large assumption to make.

1.2 Emily

This project taught me more than I ever thought it would in just about every way. On the technical side I learned first and foremost about the core of our project, ROS. I had absolutely no robotics experience going into this so it was a fantastic opportunity to learn about a whole new field and study it in depth. I also hadn't had much experience with drones before this, so it was really cool to be able to play with some high quality hardware and learn how it worked. I did have security knowledge coming in, and applied it vastly in the creation of our exploits, but in writing each one of them I gained quite a bit of new knowledge about the portion of the system with which I was working. Being able to gain all of this technical knowledge was amazing, but I think the most important learning was in the more non-technical realm.

I've always loved research and wanted to do it on a large scale, but I truly underestimated how difficult it is to do on a project longer than a term. I really had to learn (by failing time and time again) how to best manage our time and expectations. Naively I thought we would be able to accomplish a lot more, and do it much more smoothly and quickly. One of the most difficult lessons of this whole project for me was learning how to ensure that I was serving the expectations of everyone and their requirements equally, and not just focusing on what I considered to be the most important. It's difficult for most people to get serious work done on things that they don't think matter as much as something else, and I learned a great deal about trying to balance that.

On smaller scales I've often ended up managing group projects so I somewhat fell into that role in this, but it was a lot harder to juggle with this one. Time management was challenging, and keeping track of everything at once made this even more difficult, but after each term I felt I had gotten better at it. It was also clear that knowing communication is crucial and actually communicating effectively are two different things, with it being much easier said than done. If I had to say I learned the most about anything from this project, it was about communication in project management and how difficult it can be when not done perfectly.

If I had to do it all again, I would have everyone meeting much more regularly, and working together as a group in the same room. Of course having varied and busy schedules made this tough, but we could have done better.

I would also have started just about everything sooner. Humans are notoriously bad at time estimation, and this project truly proved that to me. I'm still happy with the experience I had though, because it taught me so much, from technical details to collaboration. While I may not have loved it at many points in time, I greatly appreciate my capstone experience in retrospect because I needed the challenges it provided.

1.3 Zach

I learned a lot working through this project this year, through all the ups and downs, I walked away knowing more than I did when I started. In terms of technical information, I learned the importance of a threat model in the information security field and useful it is when guiding how to approach a security evaluation of a system. I also learned a lot about drone hardware, flight controllers, the Beagle Bone Black, and of course, the Robotic Operating System. Along with ROS, I learned how to develop ROS packages, and feel much more comfortable developing for the platform. I also learned about information security methods and tools, such a fuzzers, and the role they play in trying to find vulnerabilities in a system.

In terms of what I learned on the non-technical side of things, I would say it mostly had to with how to conduct a research project of this scale. Before this project, I had never been part of an actual research project before, so having this exposer really taught me the importance of project management and communication.

With regards to project work, I learned the importance of planning, time management, team work, conflict management, and the importance of communication amongst all members and entities involved with the task at hand. Planning what needs to be done, and sticking to a productive schedule helps to keep the project on track; I did not realize how important these two things were until I found myself wishing I could go back and plan things better. It also became apparent that communication is key, and it is important to take the extra step to make sure that everyone is on the same page regarding what is going on in the project; there were many issues that came from lack of communication from all parties involved with this project, and I would say that for me, this was the biggest lesson I learned throughout all of this. Always communicate, and communicate often. With these issues, I learned how to deal with conflict and how to better listen to the input and needs of my teammates; this was the first time I really faced project conflict, so it was another important lesson during my capstone experience.

If I could go back and do it all over, I would spend much more time and care with the initial research and planning phase of the project. I feel that a lot of the problems that we ran into could have been avoided if we focused on more in-depth research and planning, especially regarding our hardware, instead of being surprised when nothing worked as expected. I would also take more care with making sure I understand what the client I asking of me and the team, and would make sure that the client and our team is committed to sticking to the goals we set during the planning stages. I would also ask for assistance and clarification much sooner, instead of continuing to try to fix things myself; not only did this waste a lot of time, but it hindered the project as a whole. Lots of lessons were learned during this experience, and I feel like I am walking away more prepared for my future.