Design Document Security for Robotics

Emily Longman, Zach Rogers, and Dominic Giacoppe

CS461 Fall Capstone

12/2/2016

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.

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. 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.

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 [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 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. [1, 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.

2.2 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. [2]

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.

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[3]. Promiscuous Mode enables us to capture wireless packets without associating with an access point. This is how a lot of wireless attacks are performed[3]. With this, we can use the Aircrack-NG suite of wireless auditing tools to attack the wireless communication channel that the drone uses[4].

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[3].

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.

3.2 Documentation of Data

The core of our project is the data that we record 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 ways we will attempt to attack the system, the data will also be quite varied. As much as I wish it were possible to have easily comparable data from all of them, it simply isn't possible. To combat this we have to try to at least document what we come up with consistently.

An article from Georgia Tech on proper research data documentation gives a few great concepts to focus on. [5] 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 we used software in manipulating, organizing, or visualizing the data though we would need to record that separately. This leads to the last point of data processing, which is also a very important one. If we change the data in any way to process it we absolutely must record that, or else the legitimacy of the final data could be ruined. Just like citing any sources we use, we have to cite anything that we do for the sake of reproducibility.

While they are options, all of these should be taken into account when we start to actually produce data. We don't necessarily need to follow the previously stated ideas as they are, and we're sure to find problems with them or better options along the way. The main concept here is that research data is precious and sensitive and we need to do our best to make our final product something that's academically valuable.

3.3 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.[6] 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[6].

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[7]. 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 Snappy Core, enabling us to use ROS[8]. 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 telemetry 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 operating distances[9]. 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, separate from what would be needed to intercept and analyze communications on the 2.4Ghz frequency[10].

What should now be clear is that there are 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.

3.3.1 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 debian install on the drone to recognize the Ethernet connection but it should be automated as part of the debian install, if it is anything like what a normal live installation media is like. The wireless connection will be more involved, but should not be any different than a regular setup of wireless connectivity for an debian system. If the need arises many guides for setting up wireless connections on linux systems exist, like this. [11] It should not be a big issue. 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.

3.4 Drone OS

For the purposes of the OS, it is not so much it's design as it is it's execution. We will be using ROS Indigo, 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 Ubuntu 14.04 for it's own operating system, will have a trivial install and setup procedure. It is detailed here[12] 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[13]. The general idea here is to flash a SD card with the given image, then get the beagle bone black to boot off it. 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[14]. 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 Debian-based Linux build for the underlying operating system on the beagle board black.

3.4.1 ROS v SROS (or rather ROS)

Once we have Ubunutu installed on both the station and debian on the drone, we can follow[12]. The gist of it is that's 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 ones for testing, but that is a question for later on as we explore our options.

4 Conclusion

4.1 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[15].

If reverse engineering proves to be unsuccessful, or difficult, we will still be in a position to attack the drone communications, using MITM style attacks, and possibly some fuzzing related attacks[16].

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[17]. 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.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[15] 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

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

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