Security for Robotics - Design Document

Emily Longman, Zach Rogers, and Dominic Giacoppe

CS461 Capstone

11/30/2016

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 find 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.

Drone OS

For the purposes of the OS, it is not so much it's design as it is it's execution. Since we are using ROS Kinetic, the most recent version of ROS, we will download and compile from source following ROS's instructions here: http://wiki.ros.org/kinetic/Installation/Source. We will need to do this twice, once on the robot itself and once on our designated control station. Since these instructions assume for the most part that the system is being built off Ubuntu, the actual challenge here will be installing Ubuntu to both systems, and in particular installing to the drone itself. The control station is a laptop and shouldn't be any harder than installing off a live cd tends to be, which means it shouldn't take more than an hour or two, assuming we use a normal laptop. Installing to the drone is a more complicated affair; the exact process is board dependent and may in fact be optional. If our drone does not need to be software wiped before we can use it, my understanding is that the prior users of our drone were already using Ubuntu so we wouldn't need to actually do anything except clean off their project-specific code. If we do need to actually build Ubuntu on the drone, Zach has compiled some documentation for our board and it would suggest that it's not difficult at all. Most of the documentation needed is found at http://elinux.org/BeagleBoardUbuntu and the general idea is that we create a live install medium on an sd card and boot off that in our drone. From there it shouldn't be too much different from a normal Ubuntu install, beyond the fact we are going for a minimal install in order to converse memory. We can also flash our OS directly to the eMMC, but doing so may permanently configure the drone, which would not be good stewardship of loaned materials. Either way, it would involve burning an ISO image to an SD card to be put in the drone, so we do not need to worry about wired/wireless configurations, just finding an SD card to burn. I do not foresee this being a particularly involved process.

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, 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 Ubuntu system. On the same page as the installation guide, about 2/3rds of the way down is a guide for configuring this as far as the software side is concerned, which should be enough for our purposes. 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.

ROS v SROS (or rather ROS)

Once we have Ubnutu installed on both the station and the drone, we can follow http://wiki.ros.org/indigo/Installation/UbuntuAR this guide for the actual installation process. 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.

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[1].

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 Snappy Core, 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 operating 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 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.

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.

Leveraging Captured Data to Develop Attack Methods

Wireshark will also assist with analying 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